AWARD NUMBER:	W81XWH-13-C-0189
TITLE:	“Virtual Environment TBI Screen (VETS)”
PRINCIPAL INVESTIGATOR:	Dr. W. Geoffrey Wright
CONTRACTING ORGANIZATION:	TEMPLE UNIVERSITY-OF THE COMMONWEALTH SYSTEM OF HIGHER EDUCATION, Philadelphia, PA 19122
REPORT DATE:	October 2014
TYPE OF REPORT:	ANNUAL REPORT
PREPARED FOR:	U.S. Army Medical Research and Materiel Command Fort Detrick, Maryland 21702-5012
DISTRIBUTION STATEMENT:	Approved for Public Release; Distribution Unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
October 2014	Annual Report	30 Sep 2013 - 29 Sep 2014

4. TITLE AND SUBTITLE

Virtual Environment TBI Screen (VETS)

5a. CONTRACT NUMBER
W81XWH-13-C-0189

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Dr. W. Geoffrey Wright, Dr. Jane McDevitt, Dr. Ryan Tierney
Max Dumont, Alex Dumont, Kwadwo Appiah-Kubi

E-Mail: wrightw@temple.edu

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Temple University – Of the Commonwealth System of Higher Education
Philadelphia, PA 19122

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The project will validate a technologically advanced, yet portable system, which will be beneficial to deployed warfighters and clinical military personnel. The device can be used to provide a quick assessment of physiological processes involved in postural control, which will assist with clinical screening of traumatic brain injury (TBI). Knowledge of acute symptoms associated with TBI can help clinicians make important decisions regarding treatment and/or return to duty. Early identification of motor symptoms caused by TBI will help expedite full recovery of service members to pre-injury health. The research plan involves a novel combination of virtual reality technology with intensive balance challenges performed on a modified Wii Balance Board. *Implementation of this device will enhance current approaches in TBI and mild TBI (i.e. concussion) diagnosis and rehabilitation management* with high relevance to the military and general population (e.g. sports concussion). The current status of the project is that a new user-interface has been designed and implemented. The device and software interface has been tested and validated relative to a high-quality research-grade forceplate and we tested clinical concurrent validity of device relative to clinical criterion-measures (Neurocom SOT and BESS). Normative values on healthy civilians and concussed individuals began testing when ORP/HRPO approval was given 2 months ago. Military test sites have been established and testing will begin as soon as HRPO approval for these sites is granted.

15. SUBJECT TERMS
Traumatic brain injury (TBI), Blast-induced TBI, concussion, balance assessment, virtual reality, field-deployable, novel technology

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			USAMRMC
Unclassified	Unclassified	Unclassified	Unclassified	129	19b. TELEPHONE NUMBER *(include area code)*

Table of Contents

Page

Project Title: "Virtual Environment TBI Screen (VETS): A field-deployable diagnostic screening system"
Contract No.: W81XWH-13-C-0189

INTRODUCTION:

The purpose of this study is to validate and test the reliability of the Virtual Environment Traumatic Brain Injury (TBI) screen (VETS) device in measuring standing balance. This system consists of software, a Wii balance board, and a large screen television that can be used to measure balance blast-concussed military personnel or in post-concussion athletes. Results from this study can help us determine if the VETS is a valid tool to help improve sensitivity and specificity to balance related changes in neurologically impaired individuals by creating an affordable, user-friendly, portable (i.e. field-deployable) device that requires minimal training and expertise to utilize. The results of this project can help reduce risk to military personnel (and athletes) who have experienced a mild TBI by accurately assessing their recovery thus reducing the likelihood that they will be prematurely returned to duty (or returned to play) (Gottshall et al 2007; Guskiewicz et al 2001). The likelihood of a second head trauma is increased during the recovery period and repeat traumas present an even greater danger than initial injury (Kelly et al 1991; Langlois et al 2006).

KEY WORDS:

Concussion, mild traumatic brain injury (mTBI), balance, Sensory Organization Test, Balance Error Scoring System, Virtual Reality, diagnosis, rehabilitation, field-deployable

PROJECT SUMMARY:

This section of the report shall describe the research accomplishments associated with each task outlined in the approved Statement of Work.

Major Task 1: Institutional Review Board Application and Approval

Subtask 1: Prepare Regulatory Documents and Research Protocol for Project

- ☑ Refine eligibility criteria, exclusion criteria, screening protocol
- ☑ Finalize consent form & human subjects protocol
- ☑ Coordinate with Sites for IRB protocol submission
- ☑ Coordinate with Sites for Temple University IRB review
- □ Coordinate with Sites for Military 2nd level IRB review (ORP/HRPO)

☑ *Milestone Achieved: Local IRB approval at Temple, GSPP, NEDU*
☑ *Milestone Achieved: HRPO approval for all civilian protocols*
□ *Milestone Initiated: HRPO approval for all military protocols*

Major Task 2: Preparation for human subject testing

Subtask1: Hiring and Training of Study Staff Advertise and interview for computer programmer

- ☑ Computer programmer hired and began optimization in October
- ☑ Advertise and interview for research assistants (RA) – Two RA's hired
- ☑ Coordinate with Sites for training study personnel on BESS and ImPACT to ensure high level of concordance among raters

☑ *Milestone Achieved: Project staff selected and trained*

Subtask 2: Validate Wii™ Balance Board relative to NeuroCom forceplate

- ☑ Running Wii Balance Board validation protocol.

☑ *Milestone Achieved: Wii Balance Board validated and ready to be integrated into VETS device*

Subtask 3: Usability optimization of VETS human-computer interface

- ☑ Computer programmer and RA's work with senior investigators to integrate new equipment and software with online analysis programs and virtual environments

☑ *Milestone Achieved: Optimized VETS device ready for validation in Task 3-5*

Major Task 3: Data Collection on healthy student population at Temple University
Subtask 1: Validate VETS relative to BESS and SOT for healthy subjects

- ☑ Implement multiple recruitment tactics including, but not limited to: online promotion of study, website links, and print advertisement for healthy volunteers
- ☑ Scheduling, and running of participants through the VETS, BESS, SOT validation protocol for healthy subjects
- ☑ Data processing and establish norms for healthy population on VETS device and BESS
- ☑ Data processing and compare to established SOT norms for healthy population

□ *Milestone On-going: Establish healthy norms for VETS, BESS, SOT*

Major Task 4: Data Collection on athlete population from Temple Concussion Program
Subtask 1: Validate VETS relative to BESS and SOT for concussed subjects

- ☑ Implement multiple recruitment tactics including, but not limited to: online promotion of study, website links, and print advertisement for healthy volunteers
- ☑ Scheduling and running of participants through the VETS, BESS validation protocol for healthy subjects
- □ Data processing and establish norms for injured population on VETS device and BESS
- □ Data processing and compare to established SOT norms for healthy population

□ *Milestone On-going: Validating VETS on injured civilian athlete population*

Major Task 5: Data Collection on military service personnel at CG Stations
Subtask 1: Validate VETS relative to BESS for healthy military subjects

- □ Recruitment of healthy military volunteers
- □ Scheduling and running of participants through the VETS and BESS protocol for healthy subjects
- □ Data processing and establish norms for healthy population on VETS device and BESS

□ *Milestone Initiated: Establish healthy norms for VETS in military population*

Major Task 6: Data Collection on military service personnel at Camp Lejeune
Subtask 1: Validate VETS relative to BESS for injured military with mTBI

- □ Recruitment of injured military with mTBI
- □ Scheduling and running of participants through the VETS and BESS protocol for healthy subjects
- □ Data processing and compare to healthy norms on VETS device and BESS

□ *Milestone Initiated: Establish healthy norms for VETS in military population*

Major Task 7: Data Analysis and Report Writing
Subtask 1: PI coordinate with Sites for monitoring data collection rates and data quality

- □ Perform all analyses using common algorithms, share output and findings with all investigators
- □ Work with each site with dissemination of findings (abstracts, presentation, publications, DOD)

□ *Milestone On-going: Report results from data analyses*

Written summary of findings

The VETS code has been written, and the system has been validated. We specifically compared our user interface with the Wii Balance Board (WBB) relative to a high-end research grade force plate (Neurocom/Natus and AMTI) and showed that we can collect data at 100 Hz. This software solution that our team designed has more capability and fidelity than any existing solution available using the WBB. A technical report is in preparation and will be submitted for publication next quarter.

Although we had our IRB approved at Temple University a year ago (Nov 2013) and immediately sent confirmation and all necessary paperwork to the (US Army Medical Research and Materiel Command (USAMRMC), Office of Research Protections (ORP), Human Research Protection Office (HRPO), presumably due to backlog in their offices, we did not get any response to our repeated inquiries into its status for six months. When we were notified of the need for minor revisions in April 2014, we quickly

addressed the requested changes, and waited for final approval which did not come until late July. We were able to begin recruiting student athletes as soon as the school semester started in September 2014.

We are actively recruiting healthy and concussed participants by flyer and advertisement, by word of mouth, and through local clinical contacts. So far this has been a successful recruitment strategy; data collection has utilized athletic college aged participants mostly from Temple University recreation and club sports. In an effort to augment recruitment and data collection we are planning to collect data with both Temple and Towson University athletes. This will help increase the potential for collection of control and concussed subjects, as well as test the VETS portability. For military populations, we have a site IRB under review at the US Naval Hospitals (Camp Lejuene and San Diego Naval Hospital).

A pilot reliability study was performed to ensure consistency of data collection between assessors. The intrarater and interrater reliability ICCs and CIs for the BESS are reported in Table 1 and Table 2, respectively. The intrarater reliability ICCs for the BESS ranged from 0.59 to 0.93, while the interrater reliability ICCs ranged from 0.94 to 0.97, which is better than previously reported (Finnoff, 2009).

Table 1. Intrarater Reliability

Intrarater Assessment-Assessor 1	**ICC**	**95% CI**	Chronbach's Alpha
Total BESS	0.879	0.576-0.977	0.871
Total BESS Firm	0.735	0.071-0.950	0.707
Total BESS Foam	0.852	0.242-0.974	0.840

Intrarater Assessment-Assessor 2	**ICC**	**95% CI**	Chronbach's Alpha
Total BESS	0.932	0.761-0.987	0.941
Total BESS Firm	0.593	-0.426-0.923	0.594
Total BESS Foam	0.891	0.618-0.979	0.892

Intrarater Assessment-Assessor 3	**ICC**	**95% CI**	Chronbach's Alpha
Total BESS	0.935	0.773-0.988	0.932
Total BESS Firm	0.672	-0.147-0.938	0.686
Total BESS Foam	0.897	0.639-0.981	0.891

Table 2. Interrater Reliability

Interrater Reliability	**ICC**	**95% CI**	**Chronbach's Alpha**
Total BESS	0.965	0.805-0.994	0.932
Total BESS Firm	0.975	0.887-0.996	0.686
Total BESS Foam	0.946	0.764-0.990	0.891

We have collected VETS, sensory organization test (SOT), and balance error scoring system (BESS) data on 23 healthy participants (14 males, 9 females; 22.8 $\pm$ 4.1 years) and 3 concussed participants (2 males; 1 female; 20.0 $\pm$ 4.4 years). The means and standard deviations of the first session SOT composite score and total BESS score for the healthy participants are 81.9 $\pm$ 4.80 and 11.9 $\pm$ 3.83, respectively. Participants suffering from a concussion score worse on both their initial the SOT as well as the BESS test with an average of 80.5 $\pm$ 5.29 and 13.0 $\pm$ 3.74, respectively. This replicates previous studies (Guskiewicz et al 1997; Broglio et al 2008). The means and standard deviations of the second session (2-week follow-up; n = 10) SOT composite score and total BESS score for the healthy participants are 87.0 $\pm$ 3.16 and 9.7 $\pm$ 3.06, respectively, demonstrating a possible learning effect (Broglio et al 2009). Table 3 displays the center of pressure sway data collected thus far within the healthy and concussed populations for each of the conditions for the initial session. Table 4 presents the center of pressure velocity data collected thus far within the healthy and concussed populations for each of the conditions for the initial session. Table 5

exhibits the root mean square in the medial-lateral direction data collected thus far within the healthy and concussed populations for each of the conditions for the initial session. Table 6 presents the root mean square in the anterior-posterior direction data collected thus far within the healthy population for each of the conditions for the initial session. The concussed athletes had worse balance scores on the VETS during all of the stances within each of the conditions compared to the healthy athletes. This difference is significant ($F_{1,24}$=4.92, p=0.036) after testing only 3 concussed subjects, which highlights the sensitivity of our new device. Tables 3-6 also illustrate the VETS scores for the healthy participant's 2-week follow-up (n = 10) for the following conditions: center of pressure sway center of pressure velocity, root mean square in the medial-lateral, root mean square in the anterior-posterior, respectively. The healthy participants' performed similarly in each of the stances within all conditions between initial and second session.

Table 3. Center of Pressure Sway Area (cm^2)

Population	**Healthy**		**Concussed**	
Test Session	***Baseline***	***2 weeks***	***Baseline***	***2 weeks***
Condition	**Mean (Std Dev)**	**Mean (Std Dev)**	**Mean (Std Dev)**	**Mean (Std Dev)**
Firm				
Static, Eyes Open	1.05 (0.6)	1.26 (1.0)	8.95 (11.6)	-
Eyes Closed	1.54 (0.8)	1.43 (0.73)	6.70 (8.0)	-
Dynamic, Eyes Open	4.50 (2.4)	2.30 (1.7)	24.8 (28.9)	-
Foam				
Static, Eyes Open	3.19 (2.4)	3.54 (2.2)	10.0 (8.4)	-
Eyes Closed	16.3 (9.7)	13.0 (6.2)	29.6 (19.1)	-
Dynamic, Eyes Open	54.9 (48.5)	22.0 (10.4)	84.6 (36.7)	-

Table 4. Center of Pressure Velocity (cm/s)

Population	**Healthy**		**Concussed**	
Test Session	***Baseline***	***2 weeks***	***Baseline***	***2 weeks***
Condition	**Mean (Std Dev)**	**Mean (Std Dev)**	**Mean (Std Dev)**	**Mean (Std Dev)**
Firm				
Static, Eyes Open	2.03(0.79)	2.08 (0.95)	3.21 (1.4)	-
Eyes Closed	2.20(0.82)	2.26 (0.89)	3.10 (1.2)	-
Dynamic, Eyes Open	2.92(1.3)	2.53 (1.1)	5.04 (2.7)	-
Foam				
Static, Eyes Open	2.38(0.97)	2.30 (0.94)	3.86 (1.8)	-
Eyes Closed	4.43(2.0)	4.06 (1.4)	6.44 (2.6)	-
Dynamic, Eyes Open	8.39(4.9)	5.44 (1.9)	12.4 (5.3)	-

Table 5. Root Mean Square Medial-Lateral (cm)

Population	**Healthy**		**Concussed**	
Test Session	***Baseline***	***2 weeks***	***Baseline***	***2 weeks***
Condition	**Mean (Std Dev)**	**Mean (Std Dev)**	**Mean (Std Dev)**	**Mean (Std Dev)**
Firm				
Static, Eyes Open	0.13 (0.06)	0.13 (0.06)	0.94 (0.89)	-
Eyes Closed	0.14 (0.08)	0.12 (0.04)	0.47 (0.33)	-
Dynamic, Eyes Open	0.34 (0.15)	0.23 (0.17)	0.85 (0.63)	-
Foam				
Static, Eyes Open	0.35 (0.23)	0.41 (0.32)	0.62 (0.40)	-
Eyes Closed	0.62 (0.21)	0.59 (0.27)	0.96 (0.45)	-
Dynamic, Eyes Open	1.37 (0.81)	0.82 (0.25)	1.64 (0.56)	-

Table 6. Root Mean Square Anterior-Posterior (cm)

Population	**Healthy**		**Concussed**	
Test Session	***Baseline***	***2 weeks***	***Baseline***	***2 weeks***
Condition	**Mean (Std Dev)**	**Mean (Std Dev)**	**Mean (Std Dev)**	**Mean (Std Dev)**
Firm				
Static, Eyes Open	0.31 (0.11)	0.30 (0.12)	1.45 (1.2)	-
Eyes Closed	0.41 (0.12)	0.35 (0.06)	0.75 (0.40)	-
Dynamic, Eyes Open	0.55 (0.15)	0.39 (0.07)	0.98 (0.71)	-
Foam				
Static, Eyes Open	0.48 (0.18)	0.44 (0.20)	0.86 (0.62)	-
Eyes Closed	1.09 (0.26)	1.01 (0.19)	1.40 (0.64)	-
Dynamic, Eyes Open	1.42 (0.40)	1.11 (0.11)	1.76 (0.31)	-

PARTICIPANTS & OTHER COLLABORATING ORGANIZATIONS

W. Geoffrey Wright, PhD	Role: PI Nearest person months worked: 5 Program oversight of all aspects of project, all personnel, and liaison for all subcontracts
Ryan Tierney, PhD, ATC	Role: Co-I Nearest person months worked: 2 Coordination of concussion recruitment, sports concussion consultation
Carole Tucker, PT, PhD	Role: Co-I Nearest person months worked: 2 IRB coordination; clinical consultation; s/w and h/w interfacing
Maxim Dumont	Role: Lead Computer programmer Nearest person months worked: 5 Design and architecture of all s/w for VETS user interface
Jane McDevitt, PhD	Role: Lab Coordinator Nearest person months worked: 3 Concussion Specialist; Subject recruitment; data collection; data analysis
Alex Dumont	Role: Graduate Research Assistant Nearest person months worked: 2 Electronics maintenance; computer programming; data analysis; presentation preparation
Kwadwo Osei Appiah-Kubi, PT	Role: Graduate Research Assistant Nearest person months worked: 3 Data collection; data analysis; presentation preparation
LT Jay Haran, PhD	Role: Co-PI Nearest person months worked: 2 Oversight of military aspects of project, military liaison for subcontracts

Contract expenditures to date (as applicable):

	This Qtr/Cumulative
Personnel:	$48,216.21 / $140,337.98
Fringe Benefits:	$8,125.51 / $30,000.42
Supplies:	$1,500 / $3,288.18

	This Qtr/Cumulative
Travel:	$1,287.78 / $2,077.09
Equipment:	$66,049.22 / $66,261.20
Other:	$2,386.02 / $3,186.92

	This Qtr/Cumulative
Subtotal:	$127,564.74 / $245,151.88
Indirect Costs:	$34,136.35 / $99,109.22
Fee:	$0 / $0
Total:	$161,701.02 / $344,261.17

KEY RESEARCH ACCOMPLISHMENTS:

- Refine eligibility criteria, exclusion criteria, screening protocol
- Finalize consent form & human subjects protocol
- Local IRB approval at Temple, GSPP, NEDU
- HRPO approval for all protocols
- Computer programmer hired and began design of VETS user-interface in October 2013 (see Appendix for all programming code and subroutines).
- Coordinate with sites for training study personnel on BESS SOT ensure high level of concordance among raters
- Project staff selected - two research assistant's hired and one lab manager
- Project staff trained on all clinical and equipment protocols (see Tables 1-2)
- Wii Balance Board validated and ready to be integrated into VETS device
- Computer programmer and RA's worked with senior investigators to integrate new equipment and software with online analysis programs and virtual environments
- Optimized VETS device and validated (see Appendix for User's Guide for VETS user-interface)
- Reliability assessment of BESS
- Defined Common Data Elements (CDE) for Federal Interagency TBI Research (FITBIR) Informatics System, established multiple new CDE's for FITBIR and will begin uploading data next quarter.
- Began recruiting from multiple Temple University and Philadelphia clinical sites
- Began collecting data on VETS, BESS, Neurocom SOT, and vestibulo-ocular/oculo-motor tests, which have recently been shown to be sensitive to the effects of mTBI (Mucha et al 2014)
- Collected and analyzed data on 23 healthy subjects and 3 concussed subjects (see Tables 3-6)
- Established subcontract with co-PI (LT Jay Haran), who is outsourcing military data collections to Coast Guard and US Naval Hospitals, which will be overseen by Dr. Richard Servatius (Director, Stress and Motivated Behavior Institute - SMBI, Syracuse VAMC)
- Data collection on military populations will begin as soon as HRPO gives approval.

REPORTABLE OUTCOMES:

- Presented preliminary findings at Moss Rehabilitation and Research
- Have abstract accepted by the Society for Neuroscience to present at annual convention in November 2014 in Washington, D.C. (see poster in Appendix 3).
- At MHSRS annual military conference in Aug 2014, LT Haran and I met with Dr. Rick Servatius and CAPT Jack Tsao, where we discussed a new arm to the study that will investigate the relationship between TBI and PTSD and how sources of stress, whether military or nonmilitary experiences, affect symptoms of TBI. As part of that study, Dr. Servatius will help us lead a project that will test motor and non-motor symptoms in a sample of subjects divided into a 2x2 matrix: 1) military with PTSD, but not mTBI, 2) military with PTSD and mTBI, 3) military with mTBI, but not PTSD, and 4) those with neither mTBI nor PTSD. Data on the VETS device and the BESS test will be collected in this 2x2 sample to determine whether sensorimotor deficits are differentiable among the injured warfighters with and without mTBI. These new aims have garnered additional funds from the DoD to conduct the study. At the request of our contract specialist, the new aims are being evaluated in a revised SOW (see Appendix 4) which the Army Contracting Officer's Representative (Christopher Baker) will evaluate before any research is initiated.

CONCLUSION:

In the first year we have succeeded in accomplishing many of our Major tasks as described in the SOW. We have designed and implemented a new portable virtual reality-based postural assessment device, which is sensitive to changes in visual-vestibular processing during sensorimotor processing (Chamelian and Feinstein 2004; Dichgans et al 1972). The importance of identifying the underlying processes that delays recovery from mTBI may be related to vestibular or visuo-vestibular processing (Alsalaheen et al 2011). We have validated the fidelity of our device relative to the industry gold-standard (Neurocom SOT) and have found concurrent validity between VETS and SOT. Our most exciting preliminary result is that after only testing a few concussed subjects relative to our partially collected healthy norms, the concussed athletes were found to have significantly worse balance scores on the VETS during all of the stances within each of the conditions. This highlights the sensitivity of our device in that even with low statistical power, our device can detect balance deficits in concussed individuals.

Despite our progress, overall the project is behind schedule due to a 6 month delay by the DoD's ORP/HRPO. However, after only 6 weeks of human subject testing we have completed half of our healthy civilian cohort and have begun testing on concussed civilians. In order to evaluate external validity of the new device, we have multiple military sites in place to collect healthy and mTBI subjects and two new military/VA collaborators will help with this data collection (Dr. Rick Servatius, PhD and LT Michael Doria USCG). We requested a no-cost extension on our project to accommodate the unexpected HRPO delays, and also so that we can incorporate the new aims that the DoD has asked us to investigate. Until approved, our goal will be to abide by the current SOW and continue collecting, analyzing, and reporting data in as efficient manner as possible. We are requesting the new SOW (see Appendix 3) be evaluated for approval so we may begin that arm of the study.

We will continue with unfinished Subtasks in the next quarter. See itemized list from SOW below.

Major Task 3: Data Collection on healthy student population at Temple University
☐*Milestone On-going: Establish healthy norms for VETS, BESS, SOT*

Major Task 4: Data Collection on athlete population from Temple Concussion Program
☐*Milestone On-going: Validating VETS on injured civilian athlete population*

Major Task 5: Data Collection on military service personnel at Coast Guard Stations
☐*Milestone Initiated: Establish healthy norms for VETS in military population*

Major Task 6: Data Collection on military service personnel at Camp Lejeune
☐*Milestone Initiated: Establish healthy norms for VETS in military population*

Major Task 7: Data Analysis and Report Writing
☐*Milestone On-going: Report results from data analyses*

REFERENCES:

Alsalaheen BA, Mucha A, Morris LO, Whitney SL, Furman JM, Camiolo-Reddy CE,... Sparto PJ. (2010) Vestibular rehabilitation for dizziness and balance disorders after concussion. *J Neurol Phys Ther.* 34(2):87-93.

Broglio SP, Ferrara MS, Sopiarz K, Kelly MS. Reliable change of the sensory organization test. Clin J Sport Med. 2008 Mar;18(2):148-54.

Broglio SP, Zhu W, Sopiarz K, Park Y (2009) Generalizability Theory Analysis of Balance Error Scoring System Reliability in Healthy Young Adults. Journal of Athletic Training; 44(5):497–502

Chamelian L, Feinstein A (2004). Outcome after mild to moderate traumatic brain injury: the role of dizziness. *Arch Phys Med Rehabil* 85. 1662-1666.

Dichgans J, Held R, Young LR, Brandt T. Moving visual scenes influence the apparent direction of gravity. Science 1972;178: 1217-1219.

Finnoff, J. T., Peterson, V. J., et al. (2009). "Intrarater and interrater reliability of the Balance Error Scoring System (BESS)." PM R 1(1): 50-54

Gottshall K.R., Gray N.L., Drake A.I., et al (2007). To investigate the influence of acute vestibular impairment following mild traumatic brain injury on subsequent ability to remain on activity duty 12 months later. *Mil Med* 172. 852-858.

Guskiewicz KM, Riemann BL, Perrin DH, Nashner LM (1997). Alternative approaches to the assessment of mild head injury in athletes. Med Sci Sports Exerc. 29:S213–S221.

Guskiewicz KM, Ross SE, Marshall SW (2001) Postural Stability and Neuropsychological Deficits After Concussion in Collegiate Athletes. J Athl Train. 36(3):263-273.

Kelly JP, Nichols JS, Filley CM, Lillehei KO, Rubinstein D, Kleinschmidt-DeMasters BK. Concussion in sports. Guidelines for the prevention of catastrophic outcome. JAMA. 1991 Nov 27;266(20):2867-9.

Langlois JA, Rutland-Brown W, Wald MM. The epidemiology and impact of traumatic brain injury: a brief overview. J Head Trauma Rehabil 2006;21:375–378.

Mucha A, Collins MW, Elbin RJ, Furman JM, Troutman-Enseki C, DeWolf RM, Marchetti G, Kontos AP. A Brief Vestibular/Ocular Motor Screening (VOMS) Assessment to Evaluate Concussions: Preliminary Findings. Am J Sports Med. 2014 Oct;42(10):2479-86.

APPENDICES:

Appendix 1

Control: _____
Concuss: _____
GUID: _____
ID #: _____

VETS Concussion Form: Initial

Participant Birth Name (first, middle, last):
_____

DOB (M/D/Y): _____Sex at birth: M F

Race: _____Age:_____

City of birth:_____Country of birth_____

Today's Date: _____ Date of Injury:_____
Year in School:_____

School: _____
Sport:_____Position:_____Years at Position:_____
Medications: _____
Supplements: _____

Have you had a musculoskeletal injury in the past 6 months?	Yes ____ Dx___ No ____
Have you had any injuries to your eye in the past month?	Yes ____ Dx___ No ____
Have you had an ear infection in the past month?	Yes ____ Dx___ No ____
*Have you been diagnosed with a learning disability?	Yes ____ Dx___ No ____
*Have you been diagnosed with ADD/ADHD?	Yes ____ Dx___ No ____
Have you been diagnosed with depression?	Yes ____ Dx___ No ____
Have you been diagnosed with history of anxiety?	Yes ____ Dx___ No ____

Concussion History? **Y ___ Dx ___ N ___**	**Headache History?** **Y___ Dx ___ N ___**	√	**History of vestibular issues?** **Y___ Dx ___ N ___**	√	**History of ocular issues?** **Y___ Dx ___ N ___**	√
Previous # 1 2 3 4 5 6+	Prior Treatment for Headache (list)		Prior Treatment for vestibular issues (list)		Prior Treatment for ocular issues (list)	
Longest Symptom Duration Days ___ Weeks ___ Months ___ Years ___ Dates Of Previous Concussions _____	History of migraine headache ___ Personal ___ Family (list)_____		History of vestibular issues ___ Personal ___ Family (list)_____		History of ocular issues ___ Personal ___ Family (list)_____	

Have you been diagnosed with a balance issues?	Yes ____ Dx___ No ____
Have you been diagnosed with a motion sickness?	Yes ____ Dx___ No ____
Have you been on a normal sleep schedule?	Yes ____ No ____
Do you know what a concussion is?	Yes ____ No ____

Game or Practice: _____ Did athlete return to play immediately after incident: Y___ N___
Was the athlete wearing protective gear Y___ N___ If so, what kind_____
Injury Description:_____
_____

Location the Injury occurred:_____
TBI Type: ___ Head on Collision ___ Impact with ground __ Hit with object ___Other _____
Location of Impact: ___Frontal ___Lft Temporal ___Rt Temporal ___Lft Parietal___Rt Parietal ___Occipital ___Neck ___Indirect Force ____Body (list what aspect of body hit)_____
Other_____
2. Immediate S & S: _____
Date of 1st s/s:

LOC: Y___ N ___ How long did you experience LOC: _____ Who told you that you experienced LOC_____
Amnesia: Y___ N ___ How long did you experience amnesia_____ **Retrograde___ Antegrade _____**
Emergency Room: _____ Y _____ N **Cat Scan**: ___ Y ___ N ___ **Other Imaging** Y ___ N ___

Glasgow Coma Scale

Eye Opening Response	**Description**	**Possible Score**	**Athlete's Score**
	Spontaneous-open with blinking at baseline	4	
	Opens to verbal command, speech, or shout	3	
	Opens to pain, not applied to face	2	
	None	1	
Verbal Response	**Description**	**Possible Score**	**Athlete's Score**
	Oriented	5	
	Confused conversation, but able to answer questions	4	
	Inappropriate responses, words discernable	3	
	Incomprehensible speech	2	
	None	1	
Motor Response	**Description**	**Possible Score**	**Athlete's Score**
	Obeys command for movement	6	
	Purposeful movement to painful stimulus	5	
	withdrawals from pain	4	
	Abnormal (spastic) flexion, decorticate posture	3	
	Abnormal (spastic) flexion, decerebrate posture	2	
	None	1	
	Total possible score	**/15**	

VETS Test					
	Firm (cm)	**Foam (cm)**	**Trial 1**	**Trial 2**	**Trial 3**
Stance Width					
Eyes Open Firm					
Eyes Closed Firm					
Dynamic Firm					
Eyes Open Foam					
Eyes Closed Foam					
Dynamic Foam					

F=fall, LT= light touch/light assist, HT = heavy touch/assist

Order of Tests(Circle order)

1. VETS	**2. VETS**
SOT	**BESS**
BESS	**SOT**

Bess Test	**Initial**		**Foot (√)**
	Firm	**Foam**	
Double Leg Stance			**R_____**
Single Leg Stance			**L_____**
Tandem Stance			
Surface Total			

Convergence	**Diplopia**
Near Point Convergence	

Vestibular – Ocular Tests	**Normal**	**Abnormal**	**# of times Lost Pace**	**Full time**	**Dizzy**		**Headache**		**Nausea**	
					Before	**After**	**Before**	**After**	**Before**	**After**
OKR										
Rapid Eye Horizontal										
Smooth pursuits slow										
Smooth pursuits fast										
OKN Drum										
VOR										
Gaze Stabilization Horiz.										

Head Thrust				
Dynamic Visual Acuity	**Stationary Line #**	Moving **Line #**	**Line # differences**	

R L U R D 20/200	R U D U L 20/65	D L U R D 20/20
L D R U L 20/160	L D R L U 20/50	L U D R L 20/16
U R D L U 20/125	D R U D L 20/40	R D U R L 20/12.5
D U L D R 20/100	U L D U R 20/32	U L R D U 20/10
L R U R D 20/80	R D R L U 20/25	

King-Devick Test	**Initial**	
	Errors	**Time**
Card 1		
Trial 1		
Trial 2		
Total	**T**	**T**
Card 2		
Trial 1		
Trial 2		
Total	**T**	**T**
Card 3		
Trial 1		
Trial 2		
Total	**T**	**T**
Total Time	**T**	**T**

Card 1 Answer Key: 6-7-4-2-5—6-0-5-7-9—4-0-8-2-3—5-3-6-8-7—1-8-3-0-6—3-1-4-8-5—4-2-7-8-5—3-7-4-8-1

Card 2 Answer Key: 7-0-1-8-4—8-2-0-7-9—7-5-6-2-9—8-4-3-6-1—5-1-0-7-4—1-3-0-2-6—8-3-5-6-0—0-7-5-6-7

Card 3 Answer Key: 1-9-7-4-8—8-6-4-2-7—7-3-9-6-5—2-9-3-7-2—5-1-7-6-2—9-4-0-2-8—1-9-7-8-0—3-8-5-3-6

Appendix 2

College of Engineering
College of Health Professions
TEMPLE UNIVERSITY

Feasibility of Affordable Custom-Designed Device with Immersive Virtual Environment for Postural Assessment

[1]Alexander Dumont, [2]Kwadwo Appiah-Kubi, [3]Maxim Dumont, [1,2]W. Geoffrey Wright
Temple University, [1]College of Engineering, [2]College of Public Health, College of Science and Technology, Philadelphia PA USA

Introduction

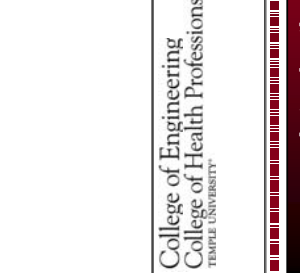
Postural stability is a complex, multisensory process, which can reflect the overall health of the sensorimotor system. Clinical assessment of balance using equipment such as the Neurocom Sensory Organization Test (SOT) focus on multisensory integration of visual, vestibular, and somatosensory inputs, which has established norms for healthy and compromised performance on these balance tests. An alternate approach that uses affordable commercially available equipment such as the Nintendo WiiBoard (WiiBB), large flat-screen television, and custom-designed software may allow clinical researchers to reliably investigate the integrity of the CNS during postural tasks.

Aims

- Compare the results in a series of objective postural assessments obtained using a custom-designed postural assessment device that emulates the conditions that have been traditionally used in the SOT and Clinical Test of Sensory Interaction and Balance (CTSIB).

Methods

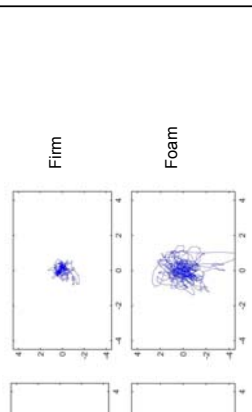
Eleven healthy adults (21-45 years) without musculoskeletal injury or known sensorimotor pathology performed 2 quite stances; on the WiiBB and on foam placed on the WiiBB, while facing a virtual environment (VE) scenes displayed on a large flatscreen television (60"). Each stance had 3 conditions; eyes closed/dark screen, eyes open/static image and eyes open/dynamic image (60 deg/s). The WiiBB recorded the COP at 30Hz for 60 sec trials. The data was analyzed using traditional quantitative measures (COP velocity, RMS, Sway Area) and non-linear measures (ApEn, SampEn, MSE).



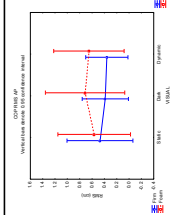
The three firm positions with eyes static, eyes closed and eyes dynamic can be seen to the left. The same thing is repeated for the foam condition, which can be seen in the lower three scenes.

Results

COP Raw Data (representative subject) shows greater instability as support surface & visual input becomes less reliable



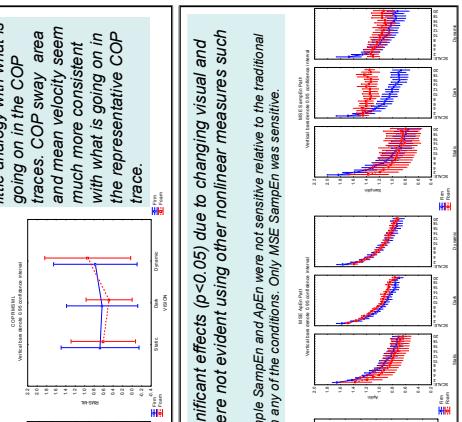
Traditional analyses rendered on the raw data yielded mixed results. While some methods showed a great amount of sensitivity to the various changing visual conditions and surfaces (COP sway area and mean velocity), other methods were less meaningful (RMS)



The RMS (Root Mean Square) revealed the least information of the traditional metrics. In both the anteroposterior and mediolateral directions, there is very little analogy with what is going on in the COP traces. COP sway area and mean velocity seem much more consistent with what is going on in the representative COP trace.

Multiscale Entropy (MSE) using SampEn shows significant effects ($p<0.05$) due to changing visual and surface conditions across temporal scales, which were not evident using other nonlinear measures such as ApEn.

Four non-linear metrics were investigated in this study. A simple SampEn and ApEn were not sensitive relative to the traditional linear metrics. MSE using ApEn did not differentiate between any of the conditions. Only MSE SampEn was sensitive.



Summary

- A custom-designed device which employs affordable, commercially-available equipment demonstrated results that replicate previous findings using research-grade medical devices
- It was sensitive to changes in postural behavior that some clinical tests cannot detect.
- Future studies will determine concurrent validity and test-retest reliability of this device relative to criterion measures and determine its sensitivity/specificity for detecting sensorimotor deficits.

Conclusion

The WiiBB turned out to be an efficient and economical alterative to the Neurocom in certain situations. However, the sampling frequency of the primitive software was not comparable to that which is built-in to the Neurocom. This has been fixed in subsequent software releases with the WiiBB attaining sampling frequency as high as the Neurocom. The metrics used in the study were comparable with that which would have been used in a Neurocom evaluation.

RMS proved to be the least informative metric. When compared with individuals COP raw data, RMS fails to be sensitive to changes across conditions. COP sway area and COP mean sway velocity were, however, very sensitive to changes in COP sway across conditions.

From the non-linear metrics, ApEn and SampEn were relatively insensitive when only a single time-scale was used, but at multiple timescales (MSE) differences were detected.

This work is supported, in part, by US Army Rapid Innovation Fund
Grant #W81XWH-13-C-0189 (RIF127023) (WGW)

Appendix 3

STATEMENT OF WORK – 09/10/2014
START DATE Sep 30, 2013

Site 1: Temple University
1301Cecil B. Moore Ave.
Philadelphia, PA 19122
PI: W. Geoffrey Wright, PhD (GW)
Co-I: Carole Tucker, PT, PhD (CT)

Site 2: Temple University Concussion Program
1800 N. Broad St.
Philadelphia, PA 19122
Co-investigator: Ryan Tierney, PhD (RT)

Site 3: US Navy Experimental Diving Unit
321 Bullfinch Rd.
Panama City, FL 32407
Co-PI: LT Jay Haran, PhD (JH)
Subcontract to RS (see site 5-6)

Site 4: Section Chief, Neurobehavioral Research Laboratory
DVA, NJHCS & Professor, Neurology & Neuroscience
Rutgers-New Jersey Medical School
Director, Stress & Motivated Behavior Institute
Collaborator: Richard Servatius, PhD (RS)

Site 5: Camp Lejeune, NC
Collaborator: Richard Servatius, PhD

Site 6: Coast Guard Stations (San Francisco, Golden Gate, Port Canaveral, St Petersburg, Seattle, Port Lauderdale, Staten Island, NYC, New Orleans)

PLEASE NOTE: Revisions to Site 4 and Site 5 are requested because availability of military subjects. We have requested permission to reassign collection of the military cohort to Dr. Servatius (RS), who has a study on warfighters with mTBI at Camp Lejeune. LT Haran will still lead this subcontract in collaboration with RS.

Abbreviations: Balance Error Scoring System = BESS; ConClin = Temple University Concussion Clinic; CG = Coast Guard,; ImPACT = Immediate Post Concussion Assessment and Cognitive Testing; NEDU = Navy Experimental Diving Unit; Navy Medical Research Center = NMRC; Sensory Organization Test = SOT; Temple = Temple University; Virtual Environment TBI Screen = VETS

Specific Aims:
This project will demonstrate the validity and reliability of a portable, field-based system, the Virtual Environment TBI Screening (VETS) device that can be used to assist with in-theater assessment of mild traumatic brain injury (mTBI, i.e. concussion). Testing will include a three-tiered approach, which will validate VETS device as a sensitive and specific TBI screening tool that generalizes across populations including military warfighters. This project will demonstrate that VETS device is a low-cost, portable screening device that is easily operated by military personnel with minimal training allowing for clinical diagnosis in less than 30 minutes, making it well-suited for field-deployment where skilled technicians and/or clinicians may not be available. Validity testing will occur as follows:

1) Test experimental concurrent validity of Wii™ Balance Board and software interface by comparing it to a high-quality research-grade forceplate
2) Test clinical concurrent validity of BESS test vs. SOT vs. VETS device on concussed individuals
3) Measure reliability change index of VETS device on healthy military cohort using test-retest differences

Projected Quarterly Enrollment

	Year 1				Year 2				Year 3				Total
Target Enrollment Validation Study (per quarter)	**Q1***	**Q2**	**Q3**	**Q4**	**Q1**	**Q2**	**Q3**	**Q4**	**Q1**	**Q2**	**Q3**	**Q4**	
Temple	-	-	-	10	10	10	10	10	-				50
ConcProgram	-	-	-	2	10	10	5	5	10	10	5	5	62
CG Stations	-	-	-	-	-	10	10	10	10	10	-	-	50
Camp Lejeune	-	-	-	-	-	12	10	10	10	10	10	-	62
Target Enrollment (cumulative)	**-**	**-**	**-**	**12**	**20**	**42**	**35**	**35**	**30**	**30**	**15**	**5**	**224**

**Q1 = Fall 2013*

*** IRB = Institutional Review Board; committee formally designated to approve, monitor, and review human subjects research*

**** HRPO = Human Research Protection Office; review and approval by HRPO office of protocols involving human subjects is required of all DoD-funded awards*

			Research Sites		
	Timeline	**%**	**Temple**	**Conc Prog**	**Lejeune & CG Sites**
Major Task 1: Institutional Review Board Application and Approval	Months				
Subtask 1: Prepare Regulatory Documents and Research Protocol for Project		-			
Refine eligibility criteria, exclusion criteria, screening protocol	1-3	11	GW/CT	RT	RS/JH
Finalize consent form & human subjects protocol	1-3	11	GW/CT	RT	
Coordinate with Sites for IRB** protocol submission	1-3	11	GW/CT	RT	RS/JH
Coordinate with Sites for Temple University IRB** review	1-6	22	GW/CT	RT	
Coordinate with Sites for Military 2nd level IRB** review (ORP/HRPO)	1-12	22	GW	RT	RS/JH
Submit amendments, adverse events and protocol deviations as needed	As Needed	-	GW	RT	RS/JH
Coordinate with Sites for annual IRB** report for continuing review	Yearly	-	GW	RT	RS/JH
*Milestone Achieved: Local IRB** approval at Temple, ConClin, NEDU*	3	11	GW/CT	RT	RS/JH
*Milestone Achieved: HRPO*** approval for all protocols and local IRB** approval through Temple.*	12	22	GW/CT	RT	RS/JH
Major Task 2: Preparation for human subject testing					
Subtask1: Hiring and Training of Study Staff					
Advertise and interview for computer programmer	1-2	5	GW		
Advertise and interview for research assistants	1-6	22	GW/CT	RT	RS/JH
Coordinate with Sites for training study personnel on BESS and ImPACT to ensure high level of concordance among raters	1-14	53	GW/CT	RT	RS/JH
Milestone Achieved: Project staff selected and trained	1-14	53	GW/CT	RT	RS/JH
Subtask 2: Validate Wii™ Balance Board relative to NeuroCom forceplate	3-7	26	GW/CT	RT	
Running Wii Balance Board validation protocol.	3-7	26	GW	RT	
Milestone Achieved: Wii Balance Board validated and ready to be integrated into VETS device	3-7	26	GW	RT	
Subtask 3: Usability optimization of VETS human-computer interface	2-10	37	GW/CT	RT	RS/JH
Computer programmer and RA's work with senior investigators to integrate new equipment and software with online analysis programs and virtual environments	2-10	37	GW		

Milestone Achieved: Optimized VETS device ready for validation in Task 3-5	5-10	37	GW	RT	RS/JH
Major Task 3: Data Collection on healthy student population at Temple University					
Subtask 1: Validate VETS relative to BESS and SOT for healthy subjects	11-24	-	GW/CT	RT	
Implement multiple recruitment tactics including, but not limited to: online promotion of study, website links, and print advertisement for healthy volunteers	11-24	90	GW/CT	RT	
Scheduling, and running of participants through the VETS, BESS, SOT validation protocol for healthy subjects	11-24	90	GW/CT	RT	
Data processing and establish norms for healthy population on VETS device and BESS	Every 10 subjects	-	GW n=50		
Data processing and compare to established SOT norms for healthy population	Every 10 subjects	-	GW n=50		
Milestone Achieved: Establish healthy norms for VETS, BESS, SOT	11-24	90	GW	RT	
Major Task 4: Data Collection on athlete population from Temple Concussion Program					
Subtask 1: Validate VETS relative to BESS and SOT for concussed subjects	11-36	-	GW/CT	RT	
Implement multiple recruitment tactics including, but not limited to: online promotion of study, website links, and print advertisement for healthy volunteers	11-36	100	GW	RT	
Scheduling, and running of participants through the VETS, BESS validation protocol for healthy subjects	11-36	100	GW	RT	
Data processing and establish norms for injured population on VETS device and BESS	Every 10 subjects	-	GW	RT	
Data processing and compare to established SOT norms for healthy population	Every 10 subjects	-	GW n=62	RT n=62	
Milestone Achieved: VETS validated on injured civilian athlete population	11-36	100	GW	RT	
Major Task 5: Data Collection on military service personnel at CG Stations					
Subtask 1: Validate VETS relative to BESS and SOT for healthy military subjects	12-36	-	GW		RS/JH
Recruitment of healthy military volunteers.	12-36	100			RS/JH
Scheduling, and running of participants through the VETS and BESS protocol for healthy subjects.	12-36	100			RS/JH
Data processing and establish norms for healthy population on VETS device and BESS.	Every 10 subjects	-			RS/JH N=50
Milestone Achieved: Establish healthy norms for VETS in military population	12-36	100	GW		RS/JH
Major Task 6: Data Collection on military service personnel at Camp Lejeune					
Subtask 1: Validate VETS relative to BESS for injured military with mTBI	12-36	-	GW		RS
Recruitment of injured military with mTBI.	12-36	100			RS
Scheduling, and running of participants through the VETS and BESS validation protocol for healthy subjects.	12-36	100			RS
Data processing and establish norms for healthy population on VETS device and BESS.	Every 10 subjects	-			RS
Milestone Achieved: VETS validated on injured military population	12-36	100	GW		RS
Major Task 7: Data Analysis and Report Writing					
Subtask 1: PI coordinate with Sites for monitoring data collection rates and data quality	11-36	-	GW/CT	RT	RS/JH
Perform all analyses using common algorithms, share output and findings with all investigators	11-36	100	GW/CT	RT	RS/JH
Work with each site with dissemination of findings (abstracts, presentation, publications, DOD)	18-36	100	GW/CT	RT	RS/JH
Milestone Achieved: Report results from data analyses	20-36	100	GW/CT	RT	RS/JH

Additional Specific Aims

Posttraumatic stress disorder (PTSD) is a major mental health problem for active military and veterans. The sources of stress, whether military or nonmilitary experiences are, and their relationship to individual differences may provide important insights toward adjustment.

1) Collect in Coast Guard, USN, and USMC personnel at various stations scale data of PTSD focusing on military (PCLM) and nonmilitary (PCLNM) experiences.
2) Compare the degree of symptom clusters expressed in PCLM and PCLNM.
3) Relate PCLM and PCLNM to individual differences (sex, marital status).
4) Compare the degree of persistence of symptom clusters in PCLM and PCLNM.

	Timeline	**NHCL/NHSD**
Major Task 1: Institutional Review Board Application and Approval	Months	
Subtask 1: Prepare Regulatory Documents and Research Protocol for CG Project		
Refine eligibility criteria, exclusion criteria, screening protocol	5/3/2013	RS
Finalize consent form & human subjects protocol	5/3/2013	
Coordinate with Sites for CG IRB protocol submission	5/3/2013	RS
Coordinate with Sites for Temple University IRB** review	1-2	GW/RS
Coordinate with Sites for Military 2nd level IRB** review (ORP/HRPO)	2-3	GW/RS
Submit amendments, adverse events and protocol deviations as needed	As Needed	RS
Coordinate with Sites for annual IRB report for continuing review	Yearly	RS
Milestone Achieved: CG IRB; VA R&D	6/3/2013	RS
Subtask 2: Prepare Regulatory Documents and Research Protocol for NHCL Project		
Refine eligibility criteria, exclusion criteria, screening protocol	5/1/2012	RS
Finalize consent form & human subjects protocol	6/1/2012	RS
Coordinate with NHCL IRB protocol submission	7/1/2012	RS
Coordinate with Sites for Temple University IRB** review	2-3	GW/RS
Coordinate with Sites for Military 2nd level IRB** review (ORP/HRPO)	3-4	GW/RS
Submit amendments, adverse events and protocol deviations as needed	As Needed	RS
Coordinate with Sites for annual IRB report for continuing review	Yearly	RS
Subtask 3: Prepare Regulatory Documents and Research Protocol for NHSD Project		

Refine eligibility criteria, exclusion criteria, screening protocol	3	RS
Finalize consent form & human subjects protocol	4	RS
Coordinate with NHCL IRB protocol submission	4	RS
Coordinate with Sites for Temple University IRB** review	5-6	GW/RS
Coordinate with Sites for Military 2nd level IRB** review (ORP/HRPO)	5-6	GW/RS
Submit amendments, adverse events and protocol deviations as needed	As Needed	RS
Coordinate with Sites for annual IRB report for continuing review	Yearly	RS
Major Task 2: Data Collection	Months	
NHCL	5-14	RS
NCSD	7-14	RS
Station Seattle	3	RS
Station Canaveral	3	RS
Station St Petersburg	4	RS
Station NYC	5	RS
Station Port Lauderdale	5	RS
Station New Orleans	6	RS
Station SanFran	6	RS
Station Golden Gate	7	RS
Station Seattle	8	RS
Station Canaveral	9	RS
Station St Petersburg	10	RS
Station NYC	11	RS
Station Port Lauderdale	11	RS
Station New Orleans	12	RS
Station SanFran	12	RS
Station Golden Gate	13	RS
Station Seattle	14	RS
Major Task 3: Data Analysis and Report Writing		
Subtask 1: PI coordinate with Sites for monitoring data collection rates and data quality	3-14	RS
Perform all analyses using common algorithms, share output and findings with all investigators	14-24	RS
Work with each site with dissemination of findings (abstracts, presentation, publications, DOD)	12-24	RS
Milestone Achieved: Report results from data analyses	24	RS

Projected Quarterly Enrollment

	Year 2				Year 3				Total
Target Enrollment Validation Study (per quarter)	**Q1***	**Q2**	**Q3**	**Q4**	**Q1**	**Q2**	**Q3**	**Q4**	
CG Stations	25	25	25	25	25	25	25	25	200
NHCL (Camp Lejeune/SanDiego)	10	20	20	20	20	20	10	-	100
NHSD	10	20	20	20	20	20	10	-	100
Target Enrollment (cumulative)	**45**	**65**	**65**	**65**	**65**	**65**	**45**	**25**	**400**

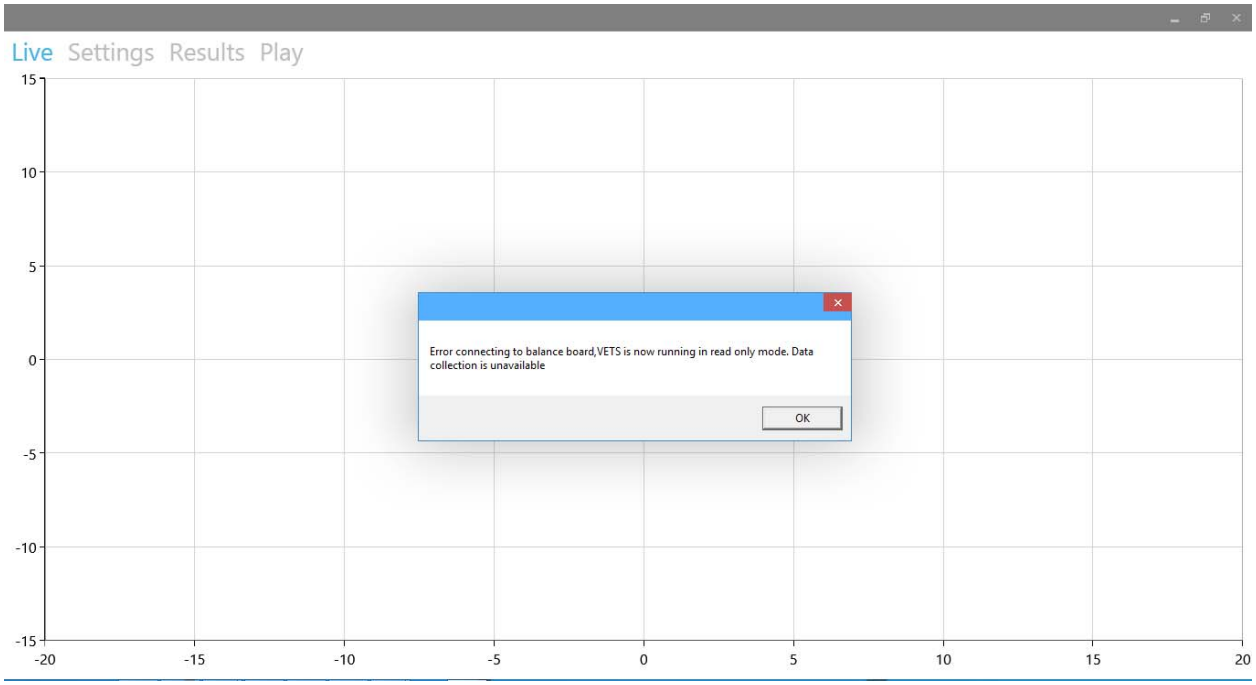
**Q1 = Fall 2014*

Appendix 4

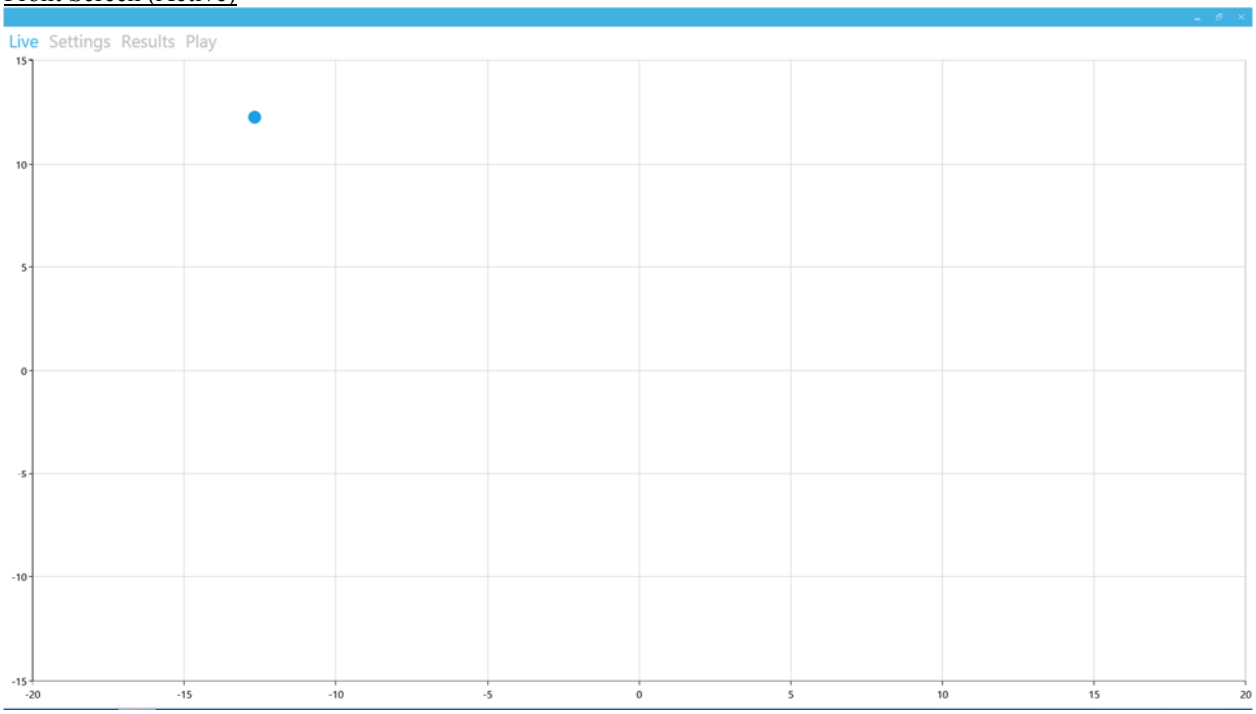
VETS USER-INTERFACE

Front Screen (Inactive)

When the balance board fails, a message box appears warning the user that the program is running in a demo mode.

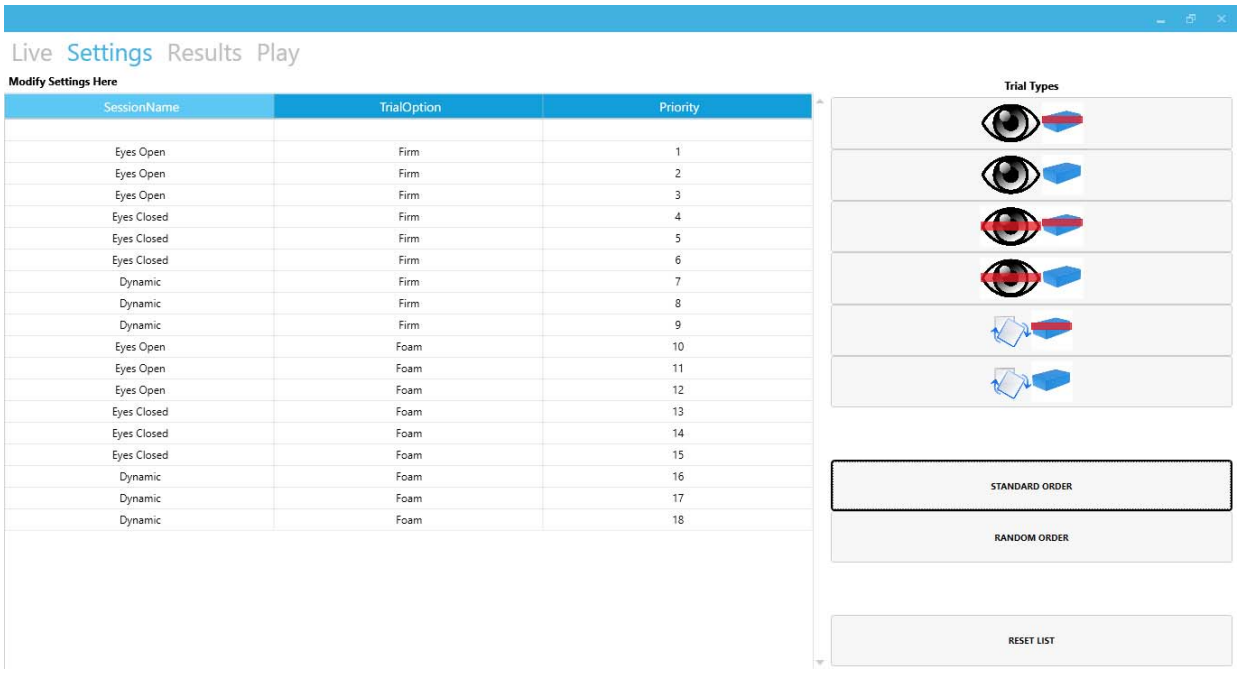


Front Screen (Active)



Settings Screen

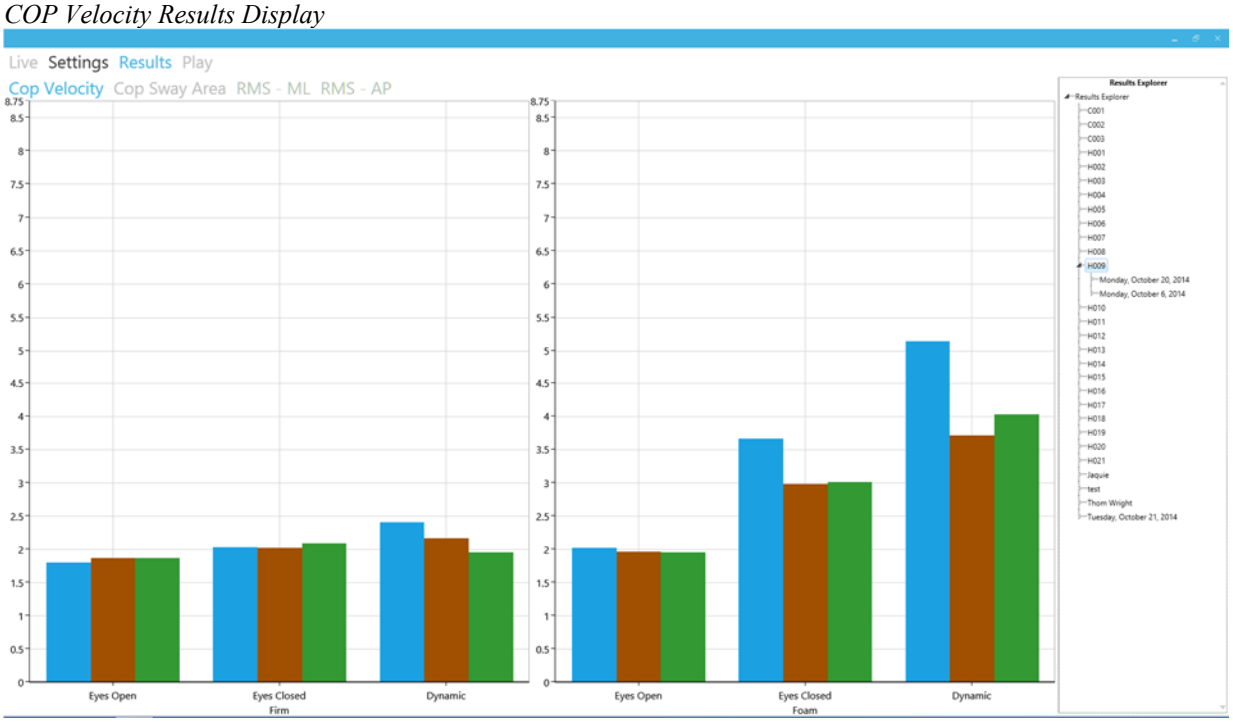
Used to select trials either one-by-one, with a single-click "Standard order" (18 trials in a standard order), with a single-click random order (18 trials in a random order), and a reset list button.


Live Settings Results Play
Modify Settings Here
SessionName
TrialOption
Priority
Eyes Open Firm 1
Eyes Open Firm 2
Eyes Open Firm 3
Eyes Closed Firm 4
Eyes Closed Firm 5
Eyes Closed Firm 6
Dynamic Firm 7
Dynamic Firm 8
Dynamic Firm 9
Eyes Open Foam 10
Eyes Open Foam 11
Eyes Open Foam 12
Eyes Closed Foam 13
Eyes Closed Foam 14
Eyes Closed Foam 15
Dynamic Foam 16
Dynamic Foam 17
Dynamic Foam 18
Trial Types
STANDARD ORDER
RANDOM ORDER
RESET LIST

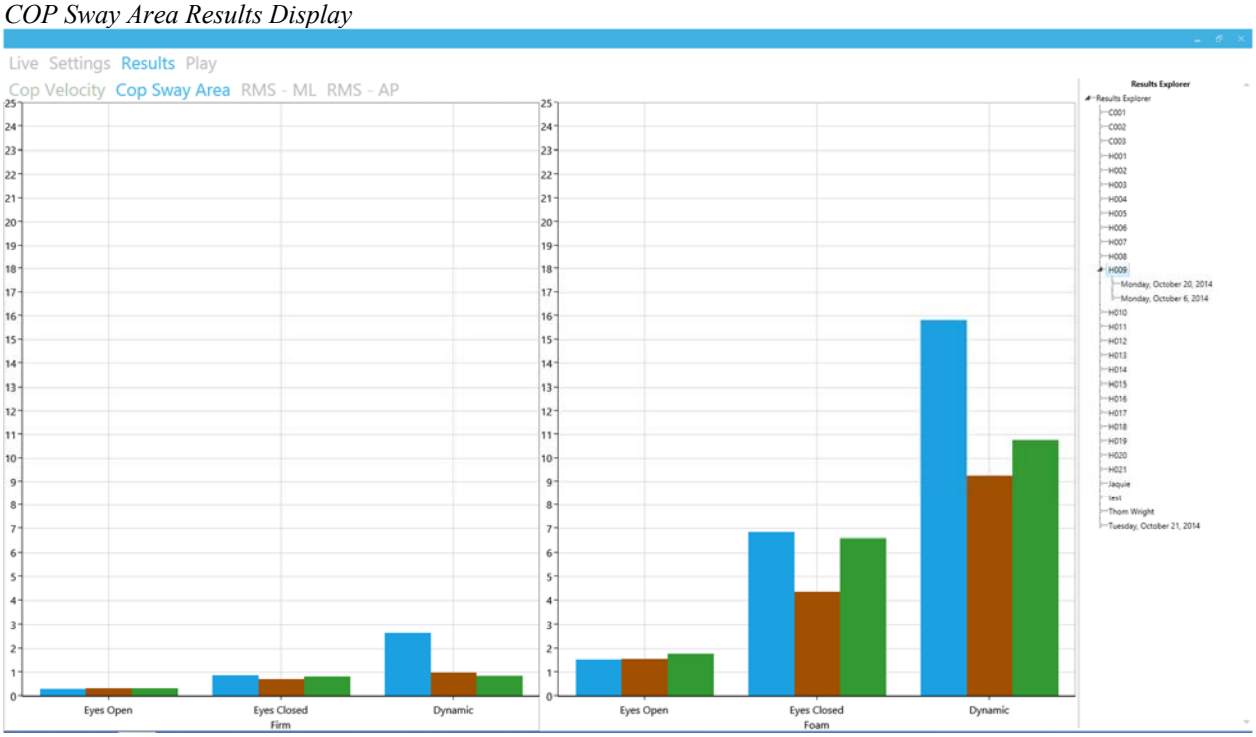
Results Screen

VETS can display 18 trials at once, with up to 3 trials per condition in color-coded format (trial 1 – blue, trial 2 – brown, trial 3 – green). Four variables are calculated from the raw COP data collected on the Wii Balance Board.

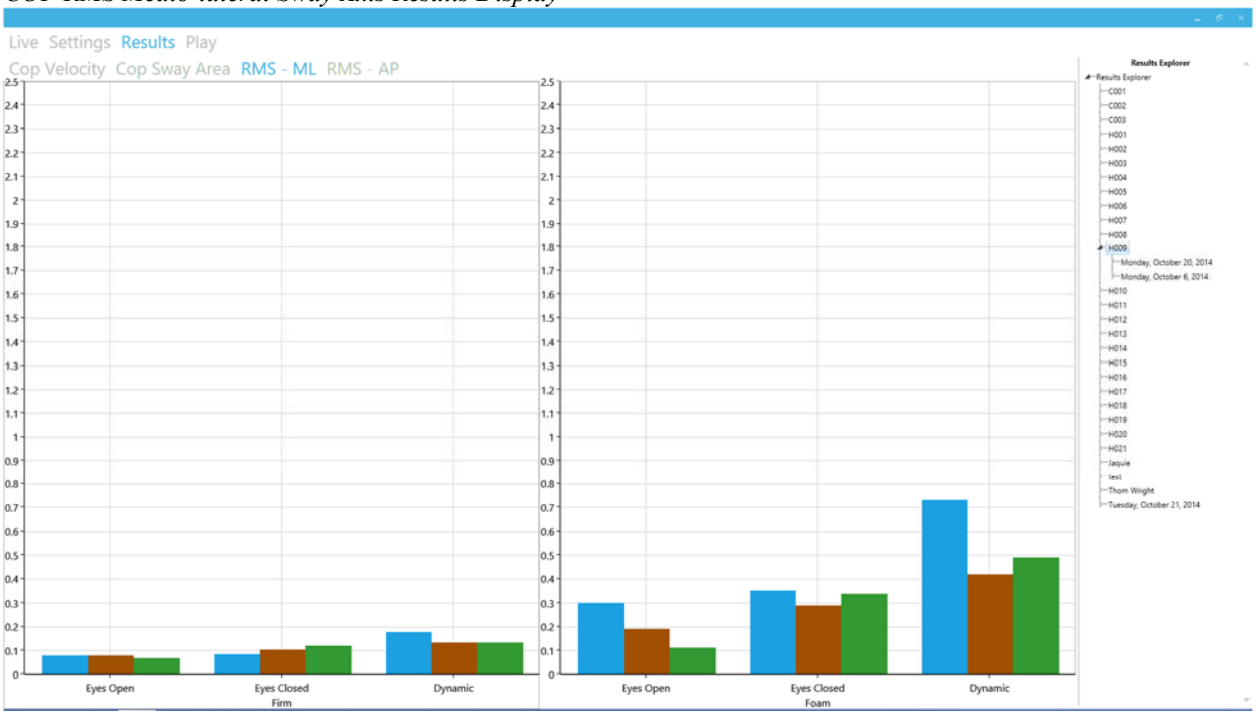
COP Velocity Results Display



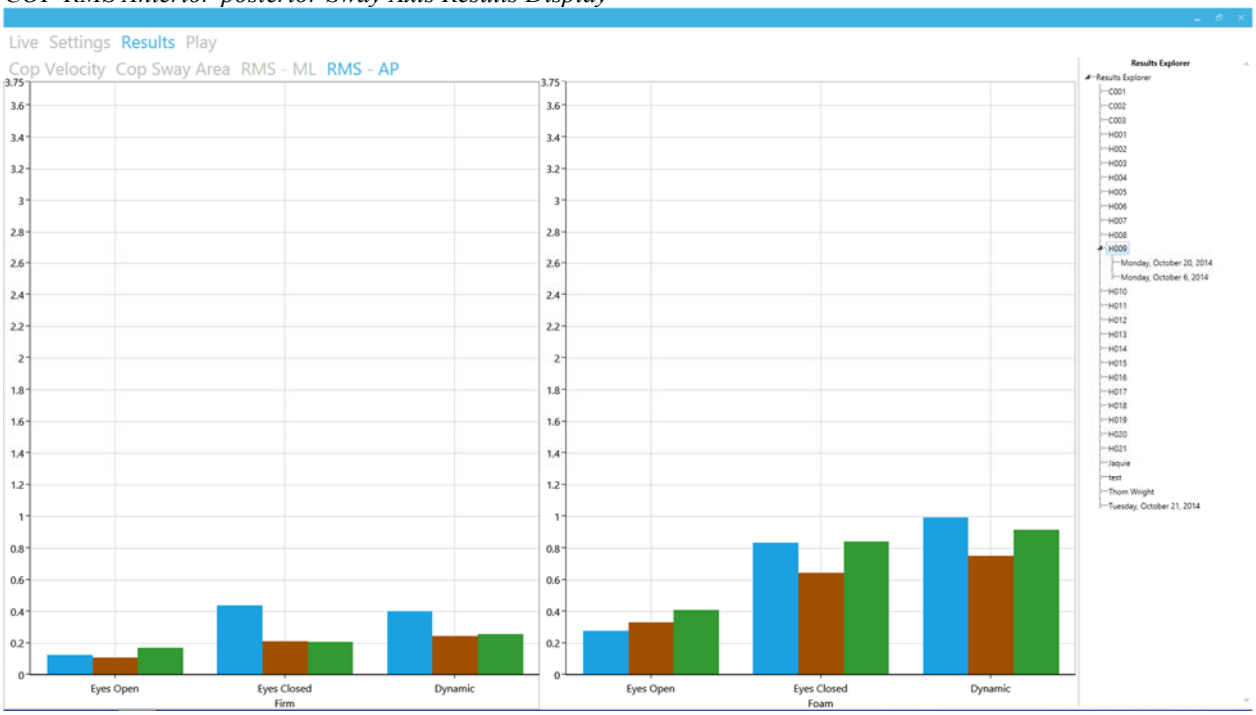
COP Sway Area Results Display



COP RMS Medio-lateral Sway Axis Results Display

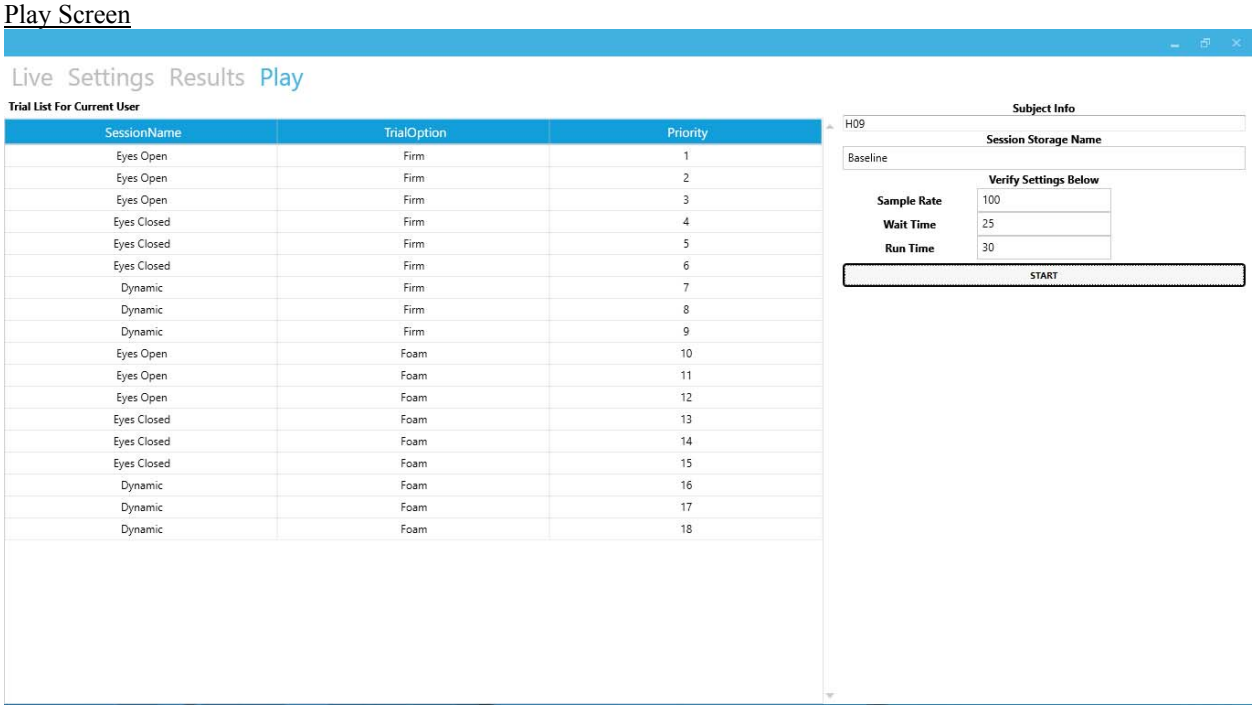


COP RMS Anterior-posterior Sway Axis Results Display



Data for each session a subject is tested in can be selected from the file tree on the right side of the screen.

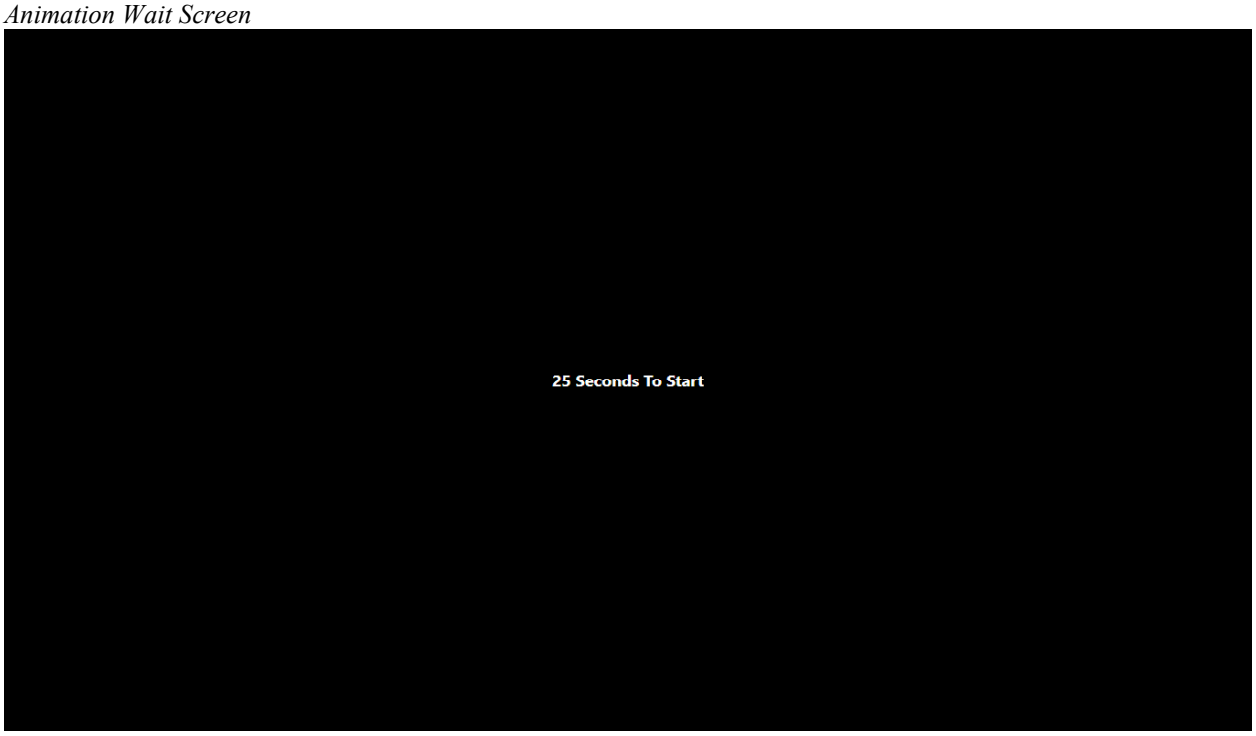
Play Screen



This screen is used to ensure all settings are correct before starting trials.

Display Screens during PLAY when COP data is collected

Animation Wait Screen



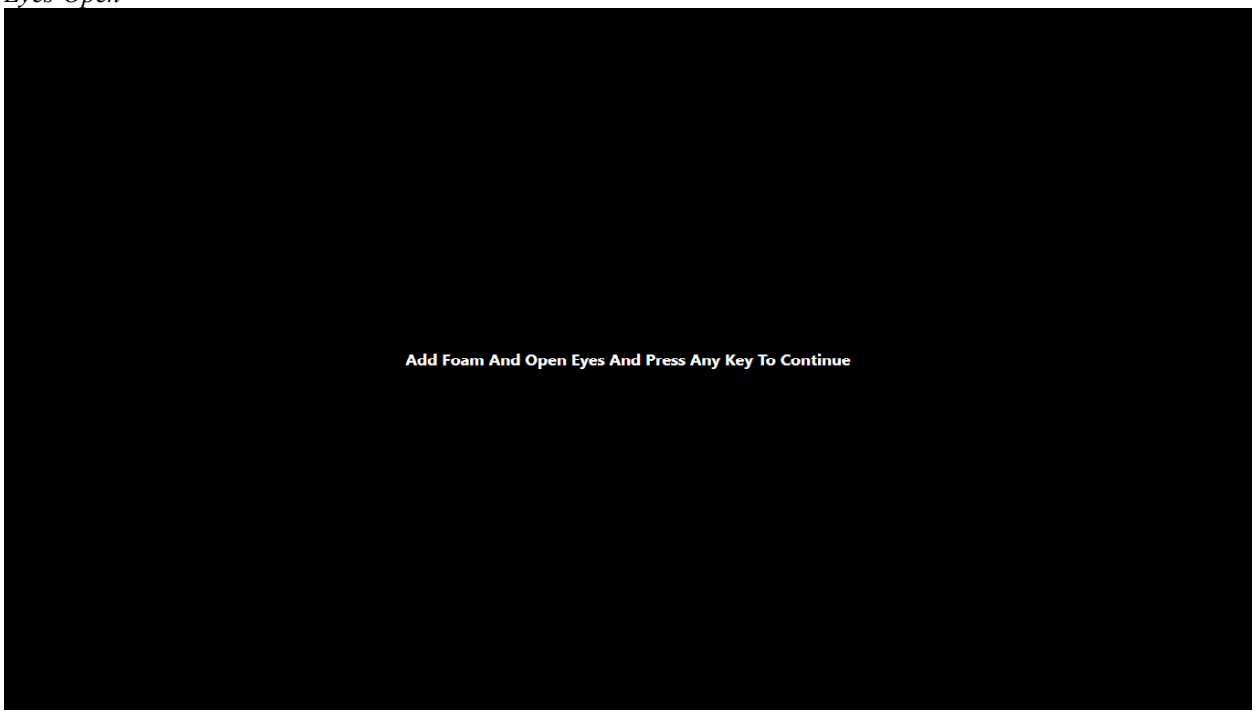
Instruction Screen displayed between trials tell experimenter/subject that next trial will be without Foam Eyes-Open

Eyes-Open Static Scene Trial

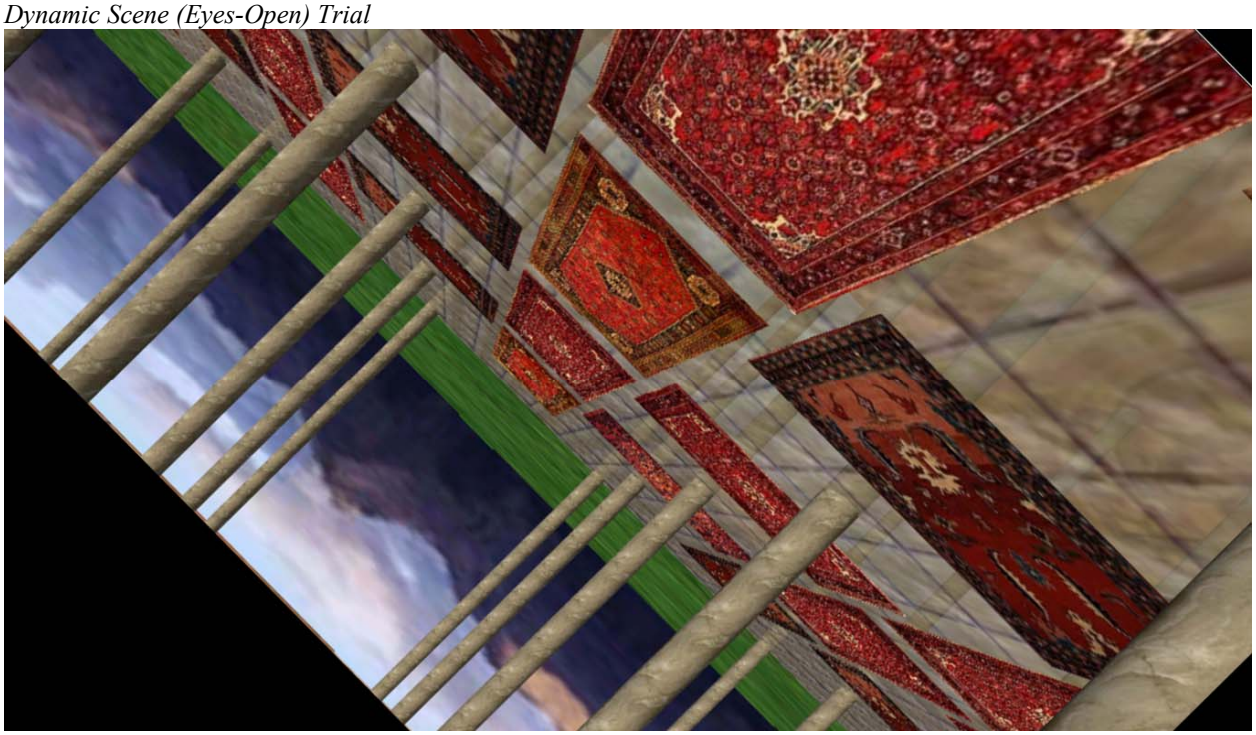
Instruction Screen displayed between trials tell experimenter/subject that next trial will be on Foam with Eyes-Closed

Eyes-Closed, Dark Screen

Instruction Screen displayed between trials tell experimenter/subject that next trial will be on Foam with Eyes-Open



Dynamic Scene (Eyes-Open) Trial



Appendix 5

Program Code and Subroutines

STARTUP FILE
Used to launch application on startup

```
<Application x:Class="VetsLegacy.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             StartupUri="Presentation/Views/MainWindow/MainWindow.xaml">
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary
Source="pack://application:,,,/MahApps.Metro;component/Styles/Controls.xaml" />
                <ResourceDictionary
Source="pack://application:,,,/MahApps.Metro;component/Styles/Fonts.xaml" />
                <ResourceDictionary
Source="pack://application:,,,/MahApps.Metro;component/Styles/Colors.xaml" />
                <ResourceDictionary
Source="pack://application:,,,/MahApps.Metro;component/Styles/Accents/Blue.xaml" />
                <ResourceDictionary

Source="pack://application:,,,/MahApps.Metro;component/Styles/Accents/BaseLight.xaml" />
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

MAIN WINDOW (VIEW)
Used to render VETS as an application

```
<controls:MetroWindow x:Class="VetsLegacy.Presentation.Views.MainWindow.MainWindow"
                      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                      xmlns:controls="http://metro.mahapps.com/winfx/xaml/controls"
                      xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
                      xmlns:mainWindow1="clr-namespace:VetsLegacy.Presentation.Views.MainWindow"
                      xmlns:results="clr-namespace:VetsLegacy.Presentation.Views.Results"
                      xmlns:play="clr-namespace:VetsLegacy.Presentation.Views.Play"
                      xmlns:front="clr-namespace:VetsLegacy.Presentation.Views.Front"
                      xmlns:settings1="clr-namespace:VetsLegacy.Presentation.Views.Settings"
                      WindowStartupLocation="CenterScreen" WindowStyle="ThreeDBorderWindow"
WindowState="Maximized"
                      Closed="MainWindow_OnClosed" BorderThickness="1" BorderBrush="Black"
x:Name="MainMetroWindow">
    <controls:MetroWindow.DataContext>
        <mainWindow1:MainWindowViewModel/>
    </controls:MetroWindow.DataContext>
    <i:Interaction.Triggers>
        <i:EventTrigger EventName="Loaded">
            <i:InvokeCommandAction Command="{Binding InitializeViewModel}" />
        </i:EventTrigger>
        <i:EventTrigger EventName="Unloaded">
            <i:InvokeCommandAction Command="{Binding UnloadViewModel}" />
        </i:EventTrigger>
    </i:Interaction.Triggers>
    <controls:MetroAnimatedTabControl HorizontalAlignment="Stretch">
        <controls:MetroTabItem Header="Live" x:Name="Live">
            <front:Front />
        </controls:MetroTabItem>
        <controls:MetroTabItem Header="Settings" x:Name="Settings">
            <settings1:Settings />
        </controls:MetroTabItem>
        <controls:MetroTabItem Header="Results" x:Name="Results">
            <results:Results />
        </controls:MetroTabItem>
        <controls:MetroTabItem Header="Play">
            <play:Play/>
```

```
        </controls:MetroTabItem>
    </controls:MetroAnimatedTabControl>
</controls:MetroWindow>
```

MAIN WINDOW BACK END LOGIC (VIEW MODEL)

Used to interact with main window and provide back end logic

```
//////////////////////////////////////////////////////////////////////////////////////////////////
//
// file:mainwindowviewmodel.cs
//
// summary:     Implements the mainwindowviewmodel class
//////////////////////////////////////////////////////////////////////////////////////////////////
//

using System.ComponentModel;
using System.Runtime.CompilerServices;
using System.Windows.Controls;
using System.Windows.Input;
using Syncfusion.Windows.Shared;
using VetsLegacy.Models.ExternalWrappers.BalanceBoard;

namespace VetsLegacy.Presentation.Views.MainWindow
{

//////////////////////////////////////////////////////////////////////////////////////////////////
//
    /// <summary>   A ViewModel for the main window. </summary>
    /// <remarks>   Maxim, 4/25/2014. </remarks>

//////////////////////////////////////////////////////////////////////////////////////////////////
//

    public class MainWindowViewModel : INotifyPropertyChanged
    {
        #region Private Properties

        /// <summary>   The content. </summary>
        private UserControl _content;

        /// <summary>   The initialise. </summary>
        private ICommand _init;

        #endregion

        #region Public Properties


//////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the content. </summary>
        ///
        /// <value> The content. </value>

//////////////////////////////////////////////////////////////////////////////////////////////////
//

        public UserControl Content
        {
            get { return _content; }
            set
            {
                _content = value;
                OnPropertyChanged("Content");
            }
        }


//////////////////////////////////////////////////////////////////////////////////////////////////
//
```

```
        /// <summary>   Gets or sets the initialize view model. </summary>
        ///
        /// <value> The initialize view model. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//

        public ICommand InitializeViewModel
        {
            get { return _init; }
            set
            {
                _init = value;
                OnPropertyChanged("InitializeViewModel");
            }
        }

        public ICommand PlayCommand { get; set; }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the unload view model. </summary>
        ///
        /// <value> The unload view model. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//

        public ICommand UnloadViewModel { get; set; }

        #endregion


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Default constructor. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//

        public MainWindowViewModel()
        {
            UnloadViewModel = new DelegateCommand(ActivateUnload,o=>true);
            PlayCommand = new DelegateCommand(ActivatePreTrial,o=>true);
        }

        private void ActivatePreTrial(object state)
        {

        }

        /// <summary>   Event queue for all listeners interested in PropertyChanged events.
</summary>
        public event PropertyChangedEventHandler PropertyChanged;


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Activates the unload method. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//

        private void ActivateUnload(object state)
```

```
        {
            BalanceBoard.DisposeCurrent();
        }


//////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Executes the property changed action. </summary>

        protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (handler != null) handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

FRONT VIEW

Used to provide live mode balance board rendering

```
<UserControl xmlns:Charts="clr-
namespace:Syncfusion.UI.Xaml.Charts;assembly=Syncfusion.SfChart.WPF"
             x:Class="VetsLegacy.Presentation.Views.Front.Front"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
             xmlns:control="http://metro.mahapps.com/winfx/xaml/controls"
             xmlns:front1="clr-namespace:VetsLegacy.Presentation.Views.Front"
             mc:Ignorable="d" x:Name="RootFront"
             d:DesignHeight="300" d:DesignWidth="300">
    <UserControl.DataContext>
        <front1:FrontViewModel />
    </UserControl.DataContext>
    <i:Interaction.Triggers>
        <i:EventTrigger EventName="Loaded">
            <i:InvokeCommandAction Command="{Binding WindowLoaded}" />
        </i:EventTrigger>
        <i:EventTrigger EventName="Unloaded">
            <i:InvokeCommandAction Command="{Binding WindowUnLoaded}" />
        </i:EventTrigger>
    </i:Interaction.Triggers>
    <Grid Margin="10">
        <Charts:SfChart>
            <Charts:SfChart.PrimaryAxis>
                <Charts:NumericalAxis Minimum="-20" Maximum="20" Interval="5" />
            </Charts:SfChart.PrimaryAxis>
            <Charts:SfChart.SecondaryAxis>
                <Charts:NumericalAxis Minimum="-15" Maximum="15" Interval="5" />
            </Charts:SfChart.SecondaryAxis>
            <Charts:ScatterSeries ItemsSource="{Binding Points}" XBindingPath="CogX"
YBindingPath="CogY" />
        </Charts:SfChart>
    </Grid>
</UserControl>
```

FRONT WINDOW BACK END LOGIC

Used to provide back end logic for rendering live mode chart system

```
//////////////////////////////////////////////////////////////////////////////////////////
//
// file:viewmodels\frontviewmodel.cs
//
// summary:     Implements the frontviewmodel class
//////////////////////////////////////////////////////////////////////////////////////////
//

using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Runtime.CompilerServices;
using System.Timers;
using System.Windows;
using System.Windows.Input;
using System.Windows.Threading;
using Syncfusion.Windows.Shared;
using VetsLegacy.Models.ExternalWrappers.BalanceBoard;

namespace VetsLegacy.Presentation.Views.Front
{

////////////////////////////////////////////////////////////////////////////////////////////////////
//
    /// <summary>   A ViewModel for the front. </summary>
    /// <remarks>   Maxim, 4/25/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
    public class FrontViewModel : INotifyPropertyChanged
    {
        #region Private Properties

        /// <summary>   The balance board. </summary>
        private BalanceBoard _balanceBoard;

        /// <summary>   The current. </summary>
        private BalanceBoardPoint _current;

        /// <summary>   true if this object is open. </summary>
        private bool _isOpen;

        /// <summary>   The loaded. </summary>
        private ICommand _loaded;

        /// <summary>   The points. </summary>
        private List<BalanceBoardPoint> _points;

        /// <summary>   The render flyout. </summary>
        private ICommand _renderFlyout;

        /// <summary>   The unload. </summary>
        private ICommand _unload;

        #endregion

        #region Public Properties

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the cog x coordinate. </summary>
        /// <value> The cog x coordinate. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public double CogX { get; set; }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the cog y coordinate. </summary>
        /// <value> The cog y coordinate. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public double CogY { get; set; }
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the top left. </summary>
        /// <value> The top left. </value>

//////////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public double TopLeft { get; set; }

//////////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the top right. </summary>
        /// <value> The top right. </value>

//////////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public double TopRight { get; set; }

//////////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the bottom right. </summary>
        /// <value> The bottom right. </value>

//////////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public double BottomRight { get; set; }

//////////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the bottom left. </summary>
        /// <value> The bottom left. </value>

//////////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public double BottomLeft { get; set; }

//////////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the window un loaded. </summary>
        /// <value> The window un loaded. </value>

//////////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public ICommand WindowUnLoaded
        {
            get { return _unload; }
            set
            {
                _unload = value;
                OnPropertyChanged("WindowUnLoaded");
            }
        }

//////////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the render flyout. </summary>
        /// <value> The render flyout. </value>

//////////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public ICommand RenderFlyout
        {
            get { return _renderFlyout; }
```

```
            set
            {
                _renderFlyout = value;
                OnPropertyChanged("RenderFlyout");
            }
        }

        //////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        /// <summary>   Gets or sets a value indicating whether the flyout open. </summary>
        /// <value> true if flyout open, false if not. </value>

        //////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        public bool FlyoutOpen
        {
            get { return _isOpen; }
            set
            {
                _isOpen = value;
                OnPropertyChanged("FlyoutOpen");
            }
        }

        //////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        /// <summary>   Gets or sets the current point. </summary>
        /// <value> The current point. </value>

        //////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        public BalanceBoardPoint CurrentPoint
        {
            get { return _current; }
            set
            {
                _current = value;
                OnPropertyChanged("CurrentPoint");
            }
        }

        //////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        /// <summary>   Gets or sets the points. </summary>
        /// <value> The points. </value>

        //////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        public List<BalanceBoardPoint> Points
        {
            get { return _points; }
            set
            {
                _points = value;
                OnPropertyChanged("Points");
            }
        }

        //////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        /// <summary>   Gets or sets the window loaded. </summary>
        /// <value> The window loaded. </value>

        //////////////////////////////////////////////////////////////////////////////////////////////////////
        //
```

```
        public ICommand WindowLoaded
        {
            get { return _loaded; }
            set
            {
                _loaded = value;
                OnPropertyChanged("WindowLoaded");
            }
        }

        #endregion

//////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private Timer _timer;

        /// <summary>   Default constructor. </summary>
        /// <remarks>   Maxim, 4/16/2014. </remarks>

//////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public FrontViewModel()
        {
            Points = new List<BalanceBoardPoint>();
            WindowLoaded = new DelegateCommand(RunInitialLoad, o => true);
            WindowUnLoaded = new DelegateCommand(ActivateUnloadMethod, o => true);
        }

//////////////////////////////////////////////////////////////////////////////////////////////////////
//

        /// <summary>   Event queue for all listeners interested in PropertyChanged events.
</summary>
        public event PropertyChangedEventHandler PropertyChanged;

//////////////////////////////////////////////////////////////////////////////////////////////////////
//

//////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Activates the flyout switch. </summary>
        /// <remarks>   Maxim, 5/15/2014. </remarks>
        /// <summary>   Activates the flyout switch. </summary>
        /// <remarks>   Maxim, 5/15/2014. </remarks>

//////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void ActivateFlyoutSwitch(object state)
        {
            FlyoutOpen = !FlyoutOpen;
        }

//////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Activates the unload method. </summary>
        /// <remarks>   Maxim, 4/16/2014. </remarks>

//////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void ActivateUnloadMethod(object state)
        {
            Dispatcher.CurrentDispatcher.Invoke(DispatcherPriority.Render, (Action) delegate
            {
```

```
                try
                {
                    _timer.Stop();
                    BalanceBoard.DisposeCurrent();
                }
                catch
                {
                }
            });
        }

////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Executes the initial load operation. </summary>
        /// <remarks>   Maxim, 4/16/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void RunInitialLoad(object state)
        {
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (Action) delegate
            {
                try
                {
                    _balanceBoard = BalanceBoard.GetBoard();
                    EnableLiveModeRendering(new object());
                }
                catch
                {
                }
            });
        }

        private void EnableLiveModeRendering(object state)
        {
            if (_balanceBoard.IsConnected)
            {
                Dispatcher.CurrentDispatcher.BeginInvoke((Action) delegate
                {
                    _timer = new Timer {Interval = 10};
                    _timer.Elapsed += WatchOnElapsed;
                    _timer.Start();
                });
            }
            else
            {
                MessageBox.Show(
                    "Error connecting to balance board,VETS is now running in read only mode. Data
collection is unavailable");
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Watch on elapsed. </summary>
        /// <remarks>   Maxim, 4/16/2014. </remarks>
        /// <param name="sender">           Source of the event. </param>
        /// <param name="elapsedEventArgs"> Elapsed event information. </param>

////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void WatchOnElapsed(object sender, ElapsedEventArgs elapsedEventArgs)
        {
            if (!_balanceBoard.IsConnected)
            {
                _balanceBoard = new BalanceBoard();
                _balanceBoard.Connect();
```

```
            }
            Points = new List<BalanceBoardPoint>
                {
                    _balanceBoard.GetPoint()
                };
        }

        protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (handler != null) handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

SETTINGS

Used to provide user with options to choose types of trials

```
<UserControl xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
             x:Class="VetsLegacy.Presentation.Views.Settings.Settings"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:settings="clr-namespace:VetsLegacy.Presentation.Views.Settings"
             xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
             mc:Ignorable="d" x:Name="SettingsWindow"
             d:DesignHeight="300" d:DesignWidth="300">
    <UserControl.DataContext>
        <settings:SettingsViewModel x:Name="SettingsViewModel" />
    </UserControl.DataContext>
    <i:Interaction.Triggers>
        <i:EventTrigger EventName="Unloaded">
            <i:InvokeCommandAction Command="{Binding OnUnload}" />
        </i:EventTrigger>
    </i:Interaction.Triggers>
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width=".5*" />
        </Grid.ColumnDefinitions>
        <Grid Grid.Column="0">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="*" />
            </Grid.RowDefinitions>
            <Label Content="Modify Settings Here" FontWeight="Bold" />
            <ScrollViewer Grid.Row="1">
                <syncfusion:GridDataControl
                    x:Name="TrialGridDataControl"
                    AllowEdit="False"
                    AllowSelection="None"
                    AutoPopulateColumns="False"
                    AutoPopulateRelations="False"
                    ColumnSizer="Star"
                    ContextMenuOptions="Default"
                    EnableContextMenu="True"
                    IsDynamicItemsSource="True"
                    ItemsSource="{Binding Trials}"
                    NotifyPropertyChanges="True"
                    VisualStyle="Metro"
                    AllowDelete="True"
                    UpdateMode="PropertyChanged"
                    ListBoxSelectionMode="MultiExtended">
                    <i:Interaction.Triggers>
                        <i:EventTrigger EventName="ItemsSourceChanged">
                            <i:InvokeCommandAction Command="{Binding SourceUpdated}" />
                        </i:EventTrigger>
                        <i:EventTrigger EventName="SourceUpdated">
                            <i:InvokeCommandAction Command="{Binding SourceUpdated}" />
```

```
                    </i:EventTrigger>
                </i:Interaction.Triggers>
                <syncfusion:GridDataControl.VisibleColumns>
                    <syncfusion:GridDataVisibleColumn MappingName="SessionName">
                        <syncfusion:GridDataVisibleColumn.HeaderStyle>
                            <syncfusion:GridDataColumnStyle HorizontalAlignment="Center" />
                        </syncfusion:GridDataVisibleColumn.HeaderStyle>
                        <syncfusion:GridDataVisibleColumn.ColumnStyle>
                            <syncfusion:GridDataColumnStyle HorizontalAlignment="Center"
Foreground="Black" />
                        </syncfusion:GridDataVisibleColumn.ColumnStyle>
                    </syncfusion:GridDataVisibleColumn>
                    <syncfusion:GridDataVisibleColumn MappingName="TrialOption">
                        <syncfusion:GridDataVisibleColumn.HeaderStyle>
                            <syncfusion:GridDataColumnStyle HorizontalAlignment="Center" />
                        </syncfusion:GridDataVisibleColumn.HeaderStyle>
                        <syncfusion:GridDataVisibleColumn.ColumnStyle>
                            <syncfusion:GridDataColumnStyle HorizontalAlignment="Center"
Foreground="Black" />
                        </syncfusion:GridDataVisibleColumn.ColumnStyle>
                    </syncfusion:GridDataVisibleColumn>
                    <syncfusion:GridDataVisibleColumn MappingName="Priority">
                        <syncfusion:GridDataVisibleColumn.HeaderStyle>
                            <syncfusion:GridDataColumnStyle HorizontalAlignment="Center" />
                        </syncfusion:GridDataVisibleColumn.HeaderStyle>
                        <syncfusion:GridDataVisibleColumn.ColumnStyle>
                            <syncfusion:GridDataColumnStyle HorizontalAlignment="Center"
Foreground="Black" />
                        </syncfusion:GridDataVisibleColumn.ColumnStyle>
                    </syncfusion:GridDataVisibleColumn>
                </syncfusion:GridDataControl.VisibleColumns>
            </syncfusion:GridDataControl>
        </ScrollViewer>
    </Grid>
    <Grid Grid.Column="1" Margin="5">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <Label Content="Trial Types" FontWeight="Bold" HorizontalAlignment="Center" />
        <Button Grid.Row="1" x:Name="StaticFirm" Command="{Binding AddToList}"
                CommandParameter="{Binding ElementName=StaticFirm,Path=Name}">
            <StackPanel Orientation="Horizontal">
                <Image Source="/Images/Trials/EyesOpenIcon.png" />
                <Image Source="/Images/Trials/NoFoam.JPG" />
            </StackPanel>
        </Button>
        <Button Grid.Row="2" x:Name="StaticFoam" Command="{Binding AddToList}"
                CommandParameter="{Binding ElementName=StaticFoam,Path=Name}">
            <StackPanel Orientation="Horizontal">
                <Image Source="/Images/Trials/EyesOpenIcon.png" />
                <Image Source="/Images/Trials/FoamIcon.JPG" />
            </StackPanel>
        </Button>
        <Button Grid.Row="3" x:Name="BlackFirm" Command="{Binding AddToList}"
                CommandParameter="{Binding ElementName=BlackFirm,Path=Name}">
            <StackPanel Orientation="Horizontal">
                <Image Source="/Images/Trials/EyesClosedIcon.png" />
                <Image Source="/Images/Trials/NoFoam.JPG" />
```

```
                </StackPanel>
            </Button>
            <Button Grid.Row="4" x:Name="BlackFoam" Command="{Binding AddToList}"
                    CommandParameter="{Binding ElementName=BlackFoam,Path=Name}">
                <StackPanel Orientation="Horizontal">
                    <Image Source="/Images/Trials/EyesClosedIcon.png" />
                    <Image Source="/Images/Trials/FoamIcon.JPG" />
                </StackPanel>
            </Button>
            <Button Grid.Row="5" x:Name="DynamicFirm" Command="{Binding AddToList}"
                    CommandParameter="{Binding ElementName=DynamicFirm,Path=Name}">
                <StackPanel Orientation="Horizontal">
                    <Image Source="/Images/Trials/Dynamic.png" />
                    <Image Source="/Images/Trials/NoFoam.JPG" />
                </StackPanel>
            </Button>
            <Button Grid.Row="6" x:Name="DynamicFoam" Command="{Binding AddToList}"
                    CommandParameter="{Binding ElementName=DynamicFoam,Path=Name}">
                <StackPanel Orientation="Horizontal">
                    <Image Source="/Images/Trials/Dynamic.png" />
                    <Image Source="/Images/Trials/FoamIcon.JPG" />
                </StackPanel>
            </Button>
            <Button Grid.Row="8" x:Name="StandardOrder" Command="{Binding AddToList}"
                    CommandParameter="{Binding ElementName=StandardOrder,Path=Name}"
Content="Standard Order" />
            <Button Grid.Row="9" x:Name="Random18" Command="{Binding AddToList}"
                    CommandParameter="{Binding ElementName=Random18,Path=Name}" Content="Random
Order" />
            <Button Grid.Row="11" Content="Reset List" Command="{Binding ClearList}" />
        </Grid>
    </Grid>
</UserControl>
```

SETTING BACK END LOGIC

Used to provide back end logic for settings view

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// file:viewmodels\settingsviewmodel.cs
//
// summary:     Implements the settingsviewmodel class
////////////////////////////////////////////////////////////////////////////////////////////////
//

using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using System.Windows;
using System.Windows.Input;
using System.Windows.Threading;
using Microsoft.Office.Interop.Excel;
using Syncfusion.Data.Extensions;
using Syncfusion.Windows.Shared;
using Syncfusion.XlsIO.Implementation.Collections;
using Syncfusion.XPS;
using VetsLegacy.Models.User.Trial;

namespace VetsLegacy.Presentation.Views.Settings
{

////////////////////////////////////////////////////////////////////////////////////////////////
//
    /// <summary>   A ViewModel for the settings. </summary>
    /// <remarks>   Maxim, 4/25/2014. </remarks>
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
    public class SettingsViewModel : INotifyPropertyChanged
    {
        #region Private Properties

        /// <summary>   The add. </summary>
        private ICommand _add;

        /// <summary>   The clear. </summary>
        private ICommand _clear;

        /// <summary>   The delete. </summary>
        private ICommand _delete;

        /// <summary>   The render time rate. </summary>
        private double _renderTimeRate;

        /// <summary>   The sample rate. </summary>
        private double _sampleRate;

        /// <summary>   The select. </summary>
        private Trial _select;

        /// <summary>   The selected trials. </summary>
        private List<Trial> _selectedTrials;

        /// <summary>   The trials. </summary>
        private ObservableCollection<Trial> _trials;

        /// <summary>   The wait time rate. </summary>
        private double _waitTimeRate;

        #endregion

        #region Public Properties

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets a list of clears. </summary>
        ///
        /// <value> A List of clears. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public ICommand ClearList
        {
            get { return _clear; }
            set
            {
                _clear = value;
                OnPropertyChanged("ClearList");
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the selected multiple trials. </summary>
        ///
        /// <value> The selected multiple trials. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public List<Trial> SelectedMultipleTrials
        {
            get { return _selectedTrials; }
```

```
            set
            {
                _selectedTrials = value;
                OnPropertyChanged("SelectedMultipleTrials");
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the selected trial. </summary>
        ///
        /// <value> The selected trial. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public Trial SelectedTrial
        {
            get { return _select; }
            set
            {
                _select = value;
                OnPropertyChanged("SelectedTrial");
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the delete. </summary>
        ///
        /// <value> The delete. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public ICommand Delete
        {
            get { return _delete; }
            set
            {
                _delete = value;
                OnPropertyChanged("Delete");
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the rendering time trials. </summary>
        ///
        /// <value> The rendering time trials. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public double RenderingTimeTrials
        {
            get { return _renderTimeRate; }
            set
            {
                _renderTimeRate = value;
                OnPropertyChanged("RenderingTimeTrials");
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the wait time trials. </summary>
        ///
```

```
        /// <value> The wait time trials. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public double WaitTimeTrials
        {
            get { return _waitTimeRate; }
            set
            {
                _waitTimeRate = value;
                OnPropertyChanged("WaitTimeTrials");
            }
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the sample rate. </summary>
        ///
        /// <value> The sample rate. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public double SampleRate
        {
            get { return _sampleRate; }
            set
            {
                _sampleRate = value;
                OnPropertyChanged("SampleRate");
            }
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the trials. </summary>
        ///
        /// <value> The trials. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public ObservableCollection<Trial> Trials
        {
            get { return _trials; }
            set
            {
                _trials = value;
                Dispatcher.CurrentDispatcher.Invoke(ResetNumbers);
                OnPropertyChanged("Trials");
            }
        }

        private void ResetNumbers()
        {
            Console.WriteLine(@"Reset Numbers Hit");
            var counter = 1;
            foreach (var trial in Trials)
            {
                trial.Priority = counter;
                counter++;
            }
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets a list of add toes. </summary>
        ///
```

```
        /// <value> A List of add toes. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public ICommand AddToList
        {
            get { return _add; }
            set
            {
                _add = value;
                OnPropertyChanged("AddToList");
            }
        }

        #endregion

        public List<MenuItem> Options
        {
            get
            {
                return new List<MenuItem>
                {
                };
            }
        }

        public ICommand SourceUpdated
        {
            get { return new DelegateCommand(UpdateTrialNumbering, CanUpdate); }
        }

        private bool CanUpdate(object obj)
        {
            return true;
        }

        private void UpdateTrialNumbering(object obj)
        {
            ResetNumbers();
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Default constructor. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public SettingsViewModel()
        {
            Dispatcher.CurrentDispatcher.BeginInvoke((System.Action) delegate
            {
                ActivateDefaultParameter();
                InitializeTrialsToList();
                SetCommandsForView();
            });
        }

        public ICommand OnUnload
        {
            get { return new DelegateCommand(Unload, CanLoad); }
        }

        private bool CanLoad(object arg)
        {
            return true;
        }
```

```
        private void Unload(object obj)
        {
            Console.WriteLine(Trials.Count);
            var trials = Trials;
            var counter = 1;
            foreach (var trial in trials)
            {
                trial.Priority = counter;
                counter++;
            }
            Trial.Trials = new List<Trial>(trials);
        }

        /// <summary>   Event queue for all listeners interested in PropertyChanged events.
</summary>
        public event PropertyChangedEventHandler PropertyChanged;

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Sets commands for view. </summary>
        ///
        /// <remarks>   Maxim, 4/21/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void SetCommandsForView()
        {
            AddToList = new DelegateCommand(ActivateAddToListCommand, CanAdd);
            Delete = new DelegateCommand<Trial>(DeleteTrial, o => true);
            ClearList = new DelegateCommand(ClearTrialList, o => true);
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Clears the trial list described by obj. </summary>
        ///
        /// <remarks>   Maxim, 5/8/2014. </remarks>
        ///
        /// <param name="obj">  The object. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void ClearTrialList(object obj)
        {
            trials = new List<Trial>();
            Trials = new ObservableCollection<Trial>();
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Deletes the trial described by obj. </summary>
        ///
        /// <remarks>   Maxim, 5/7/2014. </remarks>
        ///
        /// <param name="obj">  The object. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void DeleteTrial(Trial obj)
        {
            Trial.Trials.RemoveAt(obj.Priority);
            Trials.RemoveAt(obj.Priority);
            ResetNumbers();
        }
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Initializes the trials to list. </summary>
        ///
        /// <remarks>   Maxim, 4/21/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void InitializeTrialsToList()
        {
            List<Trial> tempTrials = Trial.ReturnTrialsAsList();
            foreach (Trial trial in tempTrials)
            {
                trial.Delete = new DelegateCommand(ActivateDeleteCommand, CanDelete);
            }
            Trials = new ObservableCollection<Trial>(tempTrials);
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Activates the default parameters. </summary>
        ///
        /// <remarks>   Maxim, 4/24/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void ActivateDefaultParameter()
        {
            SampleRate = 25;
            WaitTimeTrials = 30;
            RenderingTimeTrials = 60;
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Determine if we can delete. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="arg">  The argument. </param>
        ///
        /// <returns>   true if we can delete, false if not. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private bool CanDelete(object arg)
        {
            return true;
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Activates the delete command described by obj. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="obj">  The object. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void ActivateDeleteCommand(object obj)
        {
        }
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Determine if we can add. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="arg">  The argument. </param>
        ///
        /// <returns>   true if we can add, false if not. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private bool CanAdd(object arg)
        {
            return true;
        }

        private List<Trial> trials = new List<Trial>();

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Activates the add to list command described by obj. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="obj">  The object. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void ActivateAddToListCommand(object obj)
        {
            if (Trials.Count == 0 || Trials.Count == 18)
            {
                Trials = new ObservableCollection<Trial>();
                trials = new List<Trial>();
            }
            switch (obj.ToString())
            {
                case "BlackFirm":
                    trials.Add(new Trial(TrialType.Black, TrialOption.Firm)
                    {
                        SessionName = "Eyes Closed",
                        Priority = trials.Count + 1
                    });
                    Trials = new ObservableCollection<Trial>(trials);
                    break;
                case "BlackFoam":
                    trials.Add(new Trial("bFoam", new DelegateCommand(ActivateDeleteCommand,
CanDelete))
                    {
                        Priority = trials.Count + 1
                    });
                    Trials = new ObservableCollection<Trial>(trials);
                    break;
                case "StaticFirm":
                    trials.Add(new Trial("sFirm", new DelegateCommand(ActivateDeleteCommand,
CanDelete))
                    {
                        Priority = trials.Count + 1
                    });
                    Trials = new ObservableCollection<Trial>(trials);
                    break;
                case "StaticFoam":
                    trials.Add(new Trial("sFoam", new DelegateCommand(ActivateDeleteCommand,
CanDelete))
                    {
                        Priority = trials.Count + 1
                    });
```

```
                    Trials = new ObservableCollection<Trial>(trials);
                    break;
                case "DynamicFirm":
                    trials.Add(new Trial("dFirm", new DelegateCommand(ActivateDeleteCommand,
CanDelete))
                    {
                        Priority = trials.Count + 1
                    });
                    Trials = new ObservableCollection<Trial>(trials);
                    break;
                case "DynamicFoam":
                    trials.Add(new Trial("dFoam", new DelegateCommand(ActivateDeleteCommand,
CanDelete))
                    {
                        Priority = trials.Count + 1
                    });
                    Trials = new ObservableCollection<Trial>(trials);
                    break;
                case "StandardOrder":
                    RenderStandardOrder();
                    break;
                case "Random18":
                    RenderRandomOrder();
                    break;
            }
        }

        private void RenderRandomOrder()
        {
            Trials = new ObservableCollection<Trial>(Trial.ReturnRandom());
        }

        private void RenderStandardOrder()
        {
            Trials = new ObservableCollection<Trial>(Trial.ReturnStandard());
        }

        protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (handler != null) handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

RESULTS

Used to show results to user

```
<UserControl xmlns:Charts="clr-
namespace:Syncfusion.UI.Xaml.Charts;assembly=Syncfusion.SfChart.WPF"
             xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
             x:Class="VetsLegacy.Presentation.Views.Results.Results"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
             xmlns:controls1="http://metro.mahapps.com/winfx/xaml/controls"
             xmlns:resultsPatientTree="clr-
namespace:VetsLegacy.Presentation.Views.Results.ResultsPatientTree"
             mc:Ignorable="d" x:Name="ResultsWindow"
             d:DesignHeight="300" d:DesignWidth="300">
    <UserControl.DataContext>
        <resultsPatientTree:TreeViewModel />
    </UserControl.DataContext>
    <Grid>
        <i:Interaction.Triggers>
            <i:EventTrigger EventName="Loaded">
                <i:InvokeCommandAction Command="{Binding LoadMostRecentPatient}"/>
            </i:EventTrigger>
```

```
        </i:Interaction.Triggers>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="Auto" />
        </Grid.ColumnDefinitions>
        <Grid Grid.Column="0">
            <controls1:MetroAnimatedTabControl>
                <controls1:MetroTabItem x:Name="copVelocity" Header="Cop Velocity">
                    <Grid>
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="*" />
                            <ColumnDefinition Width="*" />
                        </Grid.ColumnDefinitions>
                        <Charts:SfChart Grid.Column="0">
                            <Charts:SfChart.PrimaryAxis>
                                <Charts:CategoryAxis Header="Firm" />
                            </Charts:SfChart.PrimaryAxis>
                            <Charts:SfChart.SecondaryAxis>
                                <Charts:NumericalAxis Minimum="0" Maximum="{Binding
CopVelocityMaxHeight}"/>
                            </Charts:SfChart.SecondaryAxis>
                            <Charts:ColumnSeries ItemsSource="{Binding
FirmEyesOpenResultsCopVelocity}"
                                                 XBindingPath="CategoryType"
YBindingPath="PointX"/>
                            <Charts:ColumnSeries ItemsSource="{Binding
FirmEyesClosedResultsCopVelocity}"
                                                 XBindingPath="CategoryType"
YBindingPath="PointX"/>
                            <Charts:ColumnSeries ItemsSource="{Binding
FirmDynamicResultsCopVelocity}"
                                                 XBindingPath="CategoryType"
                                                 YBindingPath="PointX" />
                        </Charts:SfChart>
                        <Charts:SfChart Grid.Column="1">
                            <Charts:SfChart.PrimaryAxis>
                                <Charts:CategoryAxis Header="Foam" />
                            </Charts:SfChart.PrimaryAxis>
                            <Charts:SfChart.SecondaryAxis>
                                <Charts:NumericalAxis Minimum="0" Maximum="{Binding
CopVelocityMaxHeight}"/>
                            </Charts:SfChart.SecondaryAxis>
                            <Charts:ColumnSeries ItemsSource="{Binding
FoamEyesOpenResultsCopVelocity}"
                                                 XBindingPath="CategoryType" YBindingPath="PointX"
/>
                            <Charts:ColumnSeries ItemsSource="{Binding
FoamEyesClosedResultsCopVelocity}"
                                                 XBindingPath="CategoryType" YBindingPath="PointX"
/>
                            <Charts:ColumnSeries ItemsSource="{Binding
FoamDynamicResultsCopVelocity}"
                                                 XBindingPath="CategoryType"
                                                 YBindingPath="PointX" />
                        </Charts:SfChart>
                    </Grid>
                </controls1:MetroTabItem>
                <controls1:MetroTabItem x:Name="copSway" Header="Cop Sway Area">
                    <Grid>
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="*" />
                            <ColumnDefinition Width="*" />
                        </Grid.ColumnDefinitions>
                        <Charts:SfChart Grid.Column="0">
                            <Charts:SfChart.PrimaryAxis>
                                <Charts:CategoryAxis Header="Firm" />
                            </Charts:SfChart.PrimaryAxis>
                            <Charts:SfChart.SecondaryAxis>
```

```
                            <Charts:NumericalAxis Minimum="0" Maximum="{Binding
CopSwayPcaMaxHeight}"/>
                        </Charts:SfChart.SecondaryAxis>
                        <Charts:ColumnSeries ItemsSource="{Binding
FirmEyesOpenResultsCopSway}"
                                             XBindingPath="CategoryType" YBindingPath="PointX"
/>
                        <Charts:ColumnSeries ItemsSource="{Binding
FirmEyesClosedResultsCopSway}"
                                             XBindingPath="CategoryType" YBindingPath="PointX"
/>
                        <Charts:ColumnSeries ItemsSource="{Binding FirmDynamicResultsCopSway}"
                                             XBindingPath="CategoryType"
                                             YBindingPath="PointX" />
                    </Charts:SfChart>
                    <Charts:SfChart Grid.Column="1">
                        <Charts:SfChart.PrimaryAxis>
                            <Charts:CategoryAxis Header="Foam" />
                        </Charts:SfChart.PrimaryAxis>
                        <Charts:SfChart.SecondaryAxis>
                            <Charts:NumericalAxis Minimum="0" Maximum="{Binding
CopSwayPcaMaxHeight}"/>
                        </Charts:SfChart.SecondaryAxis>
                        <Charts:ColumnSeries ItemsSource="{Binding
FoamEyesOpenResultsCopSway}"
                                             XBindingPath="CategoryType" YBindingPath="PointX"
/>
                        <Charts:ColumnSeries ItemsSource="{Binding
FoamEyesClosedResultsCopSway}"
                                             XBindingPath="CategoryType" YBindingPath="PointX"
/>
                        <Charts:ColumnSeries ItemsSource="{Binding FoamDynamicResultsCopSway}"
                                             XBindingPath="CategoryType"
                                             YBindingPath="PointX" />
                    </Charts:SfChart>
                </Grid>
            </controls1:MetroTabItem>
            <controls1:MetroTabItem Header="RMS - ML" x:Name="MetroTabItem">
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <Charts:SfChart Grid.Column="0">
                        <Charts:SfChart.PrimaryAxis>
                            <Charts:CategoryAxis Header="Firm" />
                        </Charts:SfChart.PrimaryAxis>
                        <Charts:SfChart.SecondaryAxis>
                            <Charts:NumericalAxis Minimum="0" Maximum="{Binding
RmsMlMaxHeight}"/>
                        </Charts:SfChart.SecondaryAxis>
                        <Charts:ColumnSeries ItemsSource="{Binding FirmEyesOpenRmsMl}"
                                             XBindingPath="CategoryType" YBindingPath="PointX"
/>
                        <Charts:ColumnSeries ItemsSource="{Binding FirmEyesClosedRmsMl}"
                                             XBindingPath="CategoryType" YBindingPath="PointX"
/>
                        <Charts:ColumnSeries ItemsSource="{Binding FirmDynamicRmsMl}"
                                             XBindingPath="CategoryType"
                                             YBindingPath="PointX" />
                    </Charts:SfChart>
                    <Charts:SfChart Grid.Column="1">
                        <Charts:SfChart.PrimaryAxis>
                            <Charts:CategoryAxis Header="Foam" />
                        </Charts:SfChart.PrimaryAxis>
                        <Charts:SfChart.SecondaryAxis>
                            <Charts:NumericalAxis Minimum="0" Maximum="{Binding
RmsMlMaxHeight}"/>
                        </Charts:SfChart.SecondaryAxis>
```

```
                        <Charts:ColumnSeries ItemsSource="{Binding FoamEyesOpenRmsMl}"
                                             XBindingPath="CategoryType" YBindingPath="PointX"
/>
                        <Charts:ColumnSeries ItemsSource="{Binding FoamEyesClosedRmsMl}"
                                             XBindingPath="CategoryType" YBindingPath="PointX"
/>
                        <Charts:ColumnSeries ItemsSource="{Binding FoamDynamicRmsMl}"
                                             XBindingPath="CategoryType"
                                             YBindingPath="PointX" />
                    </Charts:SfChart>
                </Grid>
            </controls1:MetroTabItem>
            <controls1:MetroTabItem Header="RMS - AP">
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <Charts:SfChart Grid.Column="0">
                        <Charts:SfChart.PrimaryAxis>
                            <Charts:CategoryAxis Header="Firm" />
                        </Charts:SfChart.PrimaryAxis>
                        <Charts:SfChart.SecondaryAxis>
                            <Charts:NumericalAxis Minimum="0" Maximum="{Binding
RmsApMaxHeight}"/>
                        </Charts:SfChart.SecondaryAxis>
                        <Charts:ColumnSeries ItemsSource="{Binding FirmEyesOpenRmsAp}"
                                             XBindingPath="CategoryType" YBindingPath="PointX"
/>
                        <Charts:ColumnSeries ItemsSource="{Binding FirmEyesClosedRmsAp}"
                                             XBindingPath="CategoryType" YBindingPath="PointX"
/>
                        <Charts:ColumnSeries ItemsSource="{Binding FirmDynamicAp}"
                                             XBindingPath="CategoryType"
                                             YBindingPath="PointX" />
                    </Charts:SfChart>
                    <Charts:SfChart Grid.Column="1">
                        <Charts:SfChart.PrimaryAxis>
                            <Charts:CategoryAxis Header="Foam" />
                        </Charts:SfChart.PrimaryAxis>
                        <Charts:SfChart.SecondaryAxis>
                            <Charts:NumericalAxis Minimum="0" Maximum="{Binding
RmsApMaxHeight}"/>
                        </Charts:SfChart.SecondaryAxis>
                        <Charts:ColumnSeries ItemsSource="{Binding FoamEyesOpenRmsAp}"
                                             XBindingPath="CategoryType" YBindingPath="PointX"
/>
                        <Charts:ColumnSeries ItemsSource="{Binding FoamEyesClosedRmsAp}"
                                             XBindingPath="CategoryType" YBindingPath="PointX"
/>
                        <Charts:ColumnSeries ItemsSource="{Binding FoamDynamicRmsAp}"
                                             XBindingPath="CategoryType" YBindingPath="PointX"
/>
                    </Charts:SfChart>
                </Grid>
            </controls1:MetroTabItem>
        </controls1:MetroAnimatedTabControl>
    </Grid>
    <Border BorderBrush="LightGray" BorderThickness="2 0 0 0" Grid.Column="1"
            Margin="4 5 2 5">
        <ScrollViewer>
            <Grid>
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="*" />
                </Grid.RowDefinitions>
                <Grid.Resources>
                    <HierarchicalDataTemplate ItemsSource="{Binding Parent}" x:Key="Template">
```

```
                            <TextBlock Text="{Binding Header}" VerticalAlignment="Center"
HorizontalAlignment="Stretch" />
                        </HierarchicalDataTemplate>
                    </Grid.Resources>
                    <TextBlock Text="Results Explorer" Grid.Row="0" FontWeight="Bold"
HorizontalAlignment="Center" />
                    <syncfusion:TreeViewAdv ItemsSource="{Binding Parent}" Width="250"
BorderBrush="Transparent"
                                            Grid.Row="1" SelectedTreeItem="{Binding
SelectedItem,Mode=TwoWay}">
                        <i:Interaction.Triggers>
                            <i:EventTrigger EventName="Loaded">
                                <i:InvokeCommandAction Command="{Binding TreeViewLoaded}" />
                            </i:EventTrigger>
                        </i:Interaction.Triggers>
                        <syncfusion:TreeViewAdv.ItemTemplate>
                            <HierarchicalDataTemplate ItemsSource="{Binding Models}">
                                <TextBlock Text="{Binding Header}" Margin="1" />
                            </HierarchicalDataTemplate>
                        </syncfusion:TreeViewAdv.ItemTemplate>
                    </syncfusion:TreeViewAdv>
                </Grid>
            </ScrollViewer>
        </Border>
    </Grid>
</UserControl>
```

RESULTS BACK END LOGIC

Used to process back end logic for Results

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Threading;
using Microsoft.Office.Interop.Excel;
using Syncfusion.Windows.Controls.VirtualTreeView;
using Syncfusion.Windows.Forms;
using Syncfusion.Windows.Shared;
using Syncfusion.XlsIO;
using VetsLegacy.Annotations;
using VetsLegacy.Models.User;
using VetsLegacy.Models.User.Results;
using VetsLegacy.Models.User.Trial;
using TreeModel = Syncfusion.Windows.Tools.Controls.TreeModel;

namespace VetsLegacy.Presentation.Views.Results.ResultsPatientTree
{
    public class TreeViewModel : INotifyPropertyChanged
    {
        public double CopVelocityMaxHeight
        {
            get { return _copVelocityHeight; }
            set
            {
                _copVelocityHeight = value;
                OnPropertyChanged("CopVelocityMaxHeight");
            }
        }

        public double CopSwayPcaMaxHeight
        {
```

```
            get { return _copSwayPcaHeight; }
            set
            {
                _copSwayPcaHeight = value;
                OnPropertyChanged("CopSwayPcaMaxHeight");
            }
        }

        public double RmsMlMaxHeight
        {
            get { return _rmsMlPcaHeight; }
            set
            {
                _rmsMlPcaHeight = value;
                OnPropertyChanged("RmsMlMaxHeight");
            }
        }

        public double RmsApMaxHeight
        {
            get { return _rmsApcaHeight; }
            set
            {
                _rmsApcaHeight = value;
                OnPropertyChanged("RmsApMaxHeight");
            }
        }

        private double GetMaxHeightForAp()
        {
            var firmMaxEyesClosed = FirmEyesClosedRmsAp.Select(m => m.PointX).ToList().Max();
            var firmMaxEyesOpen = FirmEyesOpenRmsAp.Select(m => m.PointX).ToList().Max();
            var foamMaxEyesClosed = FoamEyesClosedRmsAp.Select(m => m.PointX).ToList().Max();
            var foamMaxEyesOpen = FoamEyesOpenRmsAp.Select(m => m.PointX).ToList().Max();
            var firmDynamic = FirmDynamicAp.Select(m => m.PointX).ToList().Max();
            var foamDynamic = FoamDynamicRmsAp.Select(m => m.PointX).ToList().Max();

            var full = new List<double>();
            if (firmMaxEyesClosed != null)
            {
                full.Add(firmMaxEyesClosed);
            }
            if (firmMaxEyesOpen != null)
            {
                full.Add(firmMaxEyesOpen);
            }
            if (foamMaxEyesClosed != null)
            {
                full.Add(foamMaxEyesClosed);
            }
            if (foamMaxEyesOpen != null)
            {
                full.Add(foamMaxEyesOpen);
            }
            if (firmDynamic != null)
            {
                full.Add(firmDynamic);
            }
            if (foamDynamic != null)
            {
                full.Add(foamDynamic);
            }
            return full.Max()*2.5;
        }

        private double GetMaximHeightForRmsMl()
        {
            var firmMaxEyesClosed = FirmEyesClosedRmsMl.Select(m => m.PointX).ToList().Max();
            var firmMaxEyesOpen = FirmEyesOpenRmsMl.Select(m => m.PointX).ToList().Max();
```

```
            var foamMaxEyesClosed = FoamEyesClosedRmsMl.Select(m => m.PointX).ToList().Max();
            var foamMaxEyesOpen = FoamEyesOpenRmsMl.Select(m => m.PointX).ToList().Max();
            var firmDynamic = FirmDynamicRmsMl.Select(m => m.PointX).ToList().Max();
            var foamDynamic = FoamDynamicRmsMl.Select(m => m.PointX).ToList().Max();
            var full = new List<double>();
            if (firmMaxEyesClosed != null)
            {
                full.Add(firmMaxEyesClosed);
            }
            if (firmMaxEyesOpen != null)
            {
                full.Add(firmMaxEyesOpen);
            }
            if (foamMaxEyesClosed != null)
            {
                full.Add(foamMaxEyesClosed);
            }
            if (foamMaxEyesOpen != null)
            {
                full.Add(foamMaxEyesOpen);
            }
            if (firmDynamic != null)
            {
                full.Add(firmDynamic);
            }
            if (foamDynamic != null)
            {
                full.Add(foamDynamic);
            }
            return full.Max()*2.5;
        }
```

```
        private double GetMaxHeightForCopSway()
        {
            var firmMaxEyesClosed = FirmEyesClosedResultsCopSway.Select(m =>
m.PointX).ToList().Max();
            var firmMaxEyesOpen = FirmEyesOpenResultsCopSway.Select(m => m.PointX).ToList().Max();
            var foamMaxEyesClosed = FoamEyesClosedResultsCopSway.Select(m =>
m.PointX).ToList().Max();
            var foamMaxEyesOpen = FoamEyesOpenResultsCopSway.Select(m => m.PointX).ToList().Max();
            var firmDynamic = FirmDynamicResultsCopSway.Select(m => m.PointX).ToList().Max();
            var foamDynamic = FoamDynamicResultsCopSway.Select(m => m.PointX).ToList().Max();
            var full = new List<double>();
            if (firmMaxEyesClosed != null)
            {
                full.Add(firmMaxEyesClosed);
            }
            if (firmMaxEyesOpen != null)
            {
                full.Add(firmMaxEyesOpen);
            }
            if (foamMaxEyesClosed != null)
            {
                full.Add(foamMaxEyesClosed);
            }
            if (foamMaxEyesOpen != null)
            {
                full.Add(foamMaxEyesOpen);
            }
            if (firmDynamic != null)
            {
                full.Add(firmDynamic);
            }
            if (foamDynamic != null)
            {
                full.Add(foamDynamic);
            }
            return full.Max()*1.25;
        }
```

```
        private double GetMaxHeightForCopVelocity()
        {
            var firmMaxEyesClosed = FirmEyesClosedResultsCopVelocity.Select(m =>
m.PointX).ToList().Max();
            var firmMaxEyesOpen = FirmEyesOpenResultsCopVelocity.Select(m =>
m.PointX).ToList().Max();
            var foamMaxEyesClosed = FoamEyesClosedResultsCopVelocity.Select(m =>
m.PointX).ToList().Max();
            var foamMaxEyesOpen = FoamEyesOpenResultsCopVelocity.Select(m =>
m.PointX).ToList().Max();
            var firmDynamic = FirmDynamicResultsCopVelocity.Select(m => m.PointX).ToList().Max();
            var foamDynamic = FoamDynamicResultsCopVelocity.Select(m => m.PointX).ToList().Max();

            var full = new List<double>();
            if (firmMaxEyesClosed != null)
            {
                full.Add(firmMaxEyesClosed);
            }
            if (firmMaxEyesOpen != null)
            {
                full.Add(firmMaxEyesOpen);
            }
            if (foamMaxEyesClosed != null)
            {
                full.Add(foamMaxEyesClosed);
            }
            if (foamMaxEyesOpen != null)
            {
                full.Add(foamMaxEyesOpen);
            }
            if (firmDynamic != null)
            {
                full.Add(firmDynamic);
            }
            if (foamDynamic != null)
            {
                full.Add(foamDynamic);
            }
            return full.Max()*1.25;
        }

        private List<Result> _firmEyesOpenResultsCopVelocity;
        private ICommand _load;
        private ICommand _setSelectedItem;

        public List<Result> FoamEyesOpenRmsAp
        {
            get { return _foamEyesOpenRmsAp; }
            set
            {
                _foamEyesOpenRmsAp = value;
                OnPropertyChanged("FoamEyesOpenRmsAp");
            }
        }

        public List<Result> FoamEyesClosedRmsAp
        {
            get { return _foamEyesClosedRmsAp; }
            set
            {
                _foamEyesClosedRmsAp = value;
                OnPropertyChanged("FoamEyesClosedRmsAp");
            }
        }

        public List<Result> FoamDynamicRmsAp
        {
            get { return _foamDynamicRmsAp; }
```

```
            set
            {
                _foamDynamicRmsAp = value;
                OnPropertyChanged("FoamDynamicRmsAp");
            }
        }

        public List<Result> FirmEyesOpenRmsAp
        {
            get { return _firmEyesOpenAp; }
            set
            {
                _firmEyesOpenAp = value;
                OnPropertyChanged("FirmEyesOpenRmsAp");
            }
        }

        public List<Result> FirmEyesClosedRmsAp
        {
            get { return _firmEyesClosedAp; }
            set
            {
                _firmEyesClosedAp = value;
                OnPropertyChanged("FirmEyesClosedRmsAp");
            }
        }

        public List<Result> FirmDynamicAp
        {
            get { return _firmDynamicAp; }
            set
            {
                _firmDynamicAp = value;
                OnPropertyChanged("FirmDynamicAp");
            }
        }

        public List<Result> FoamEyesClosedRmsMl
        {
            get { return _foamEyesClosedRmsMl; }
            set
            {
                _foamEyesClosedRmsMl = value;
                OnPropertyChanged("FoamEyesClosedRmsMl");
            }
        }

        public List<Result> FirmEyesOpenRmsMl
        {
            get { return _firmEyesOpenRmsMl; }
            set
            {
                _firmEyesOpenRmsMl = value;
                OnPropertyChanged("FirmEyesOpenRmsMl");
            }
        }

        public List<Result> FirmEyesClosedRmsMl
        {
            get { return _firmEyesClosedRmsMl; }
            set
            {
                _firmEyesClosedRmsMl = value;
                OnPropertyChanged("FirmEyesClosedRmsMl");
            }
        }

        public List<Result> FirmDynamicRmsMl
        {
```

```
            get { return _firmDynamicRmsMl; }
            set
            {
                _firmDynamicRmsMl = value;
                OnPropertyChanged("FirmDynamicRmsMl");
            }
        }

        public List<Result> FoamEyesOpenRmsMl
        {
            get { return _foamEyesOpenRmsMl; }
            set
            {
                _foamEyesOpenRmsMl = value;
                OnPropertyChanged("FoamEyesOpenRmsMl");
            }
        }

        public List<Result> FoamDynamicRmsMl
        {
            get { return _foamDynamicRmsMl; }
            set
            {
                _foamDynamicRmsMl = value;
                OnPropertyChanged("FoamDynamicRmsMl");
            }
        }

        public List<Result> FoamEyesClosedResultsCopSway
        {
            get { return _foamEyesClosedResultsCopSway; }
            set
            {
                _foamEyesClosedResultsCopSway = value;
                OnPropertyChanged("FoamEyesClosedResultsCopSway");
            }
        }

        public List<Result> FoamDynamicResultsCopSway
        {
            get { return _foamDynamicResultsCopSway; }
            set
            {
                _foamDynamicResultsCopSway = value;
                OnPropertyChanged("FoamDynamicResultsCopSway");
            }
        }

        public List<Result> FoamEyesOpenResultsCopSway
        {
            get { return _foamEyesOpenResultsCopSway; }
            set
            {
                _foamEyesOpenResultsCopSway = value;
                OnPropertyChanged("FoamEyesOpenResultsCopSway");
            }
        }

        public List<Result> FirmDynamicResultsCopSway
        {
            get { return _firmDynamicResultsCopSway; }
            set
            {
                _firmDynamicResultsCopSway = value;
                OnPropertyChanged("FirmDynamicResultsCopSway");
            }
        }

        public List<Result> FirmEyesClosedResultsCopSway
```

```
        {
            get { return _firmEyesClosedResultsCopSway; }
            set
            {
                _firmEyesClosedResultsCopSway = value;
                OnPropertyChanged("FirmEyesClosedResultsCopSway");
            }
        }

        public List<Result> FirmEyesOpenResultsCopSway
        {
            get { return _firmEyesOpenCopSway; }
            set
            {
                _firmEyesOpenCopSway = value;
                OnPropertyChanged("FirmEyesOpenResultsCopSway");
            }
        }

        public List<Result> FoamDynamicResultsCopVelocity
        {
            get { return _foamDynamicResultsCopVelocity; }
            set
            {
                _foamDynamicResultsCopVelocity = value;
                OnPropertyChanged("FoamDynamicResultsCopVelocity");
            }
        }

        public List<Result> FoamEyesClosedResultsCopVelocity
        {
            get { return _foamEyesClosedResultsCopVelocity; }
            set
            {
                _foamEyesClosedResultsCopVelocity = value;
                OnPropertyChanged("FoamEyesClosedResultsCopVelocity");
            }
        }

        public List<Result> FoamEyesOpenResultsCopVelocity
        {
            get { return _foamEyesOpenResultsCopVelocity; }
            set
            {
                _foamEyesOpenResultsCopVelocity = value;
                OnPropertyChanged("FoamEyesOpenResultsCopVelocity");
            }
        }

        public List<Result> FirmDynamicResultsCopVelocity
        {
            get { return _firmDynamicCopVelocity; }
            set
            {
                _firmDynamicCopVelocity = value;
                OnPropertyChanged("FirmDynamicResultsCopVelocity");
            }
        }

        public List<Result> FirmEyesClosedResultsCopVelocity
        {
            get { return _firmEyesClosedResultsCopVelocity; }
            set
            {
                _firmEyesClosedResultsCopVelocity = value;
                OnPropertyChanged("FirmEyesClosedResultsCopVelocity");
            }
        }
```

```
        public List<Result> FirmEyesOpenResultsCopVelocity
        {
            get { return _firmEyesOpenResultsCopVelocity; }
            set
            {
                _firmEyesOpenResultsCopVelocity = value;
                OnPropertyChanged("FirmEyesOpenResultsCopVelocity");
            }
        }

        public ICommand SetSelectedItemCommand
        {
            get { return _setSelectedItem; }
            set
            {
                _setSelectedItem = value;
                OnPropertyChanged("SetSelectedItemCommand");
            }
        }

        public Models.TreeModel SelectedItem
        {
            set { RenderSelectedItemValue(value); }
        }

        private void RenderSelectedItemValue(Models.TreeModel value)
        {
            if (!Directory.Exists(value.PathToItem))
            {
                return;
            }
            var files = Directory.GetFiles(value.PathToItem, "*.xlsx",
SearchOption.AllDirectories);
            var firmEyesOpenFiles = files.Where(m => m.Contains("Firm") && m.Contains("Eyes
Open"));
            var firmEyesOpenCopVelocity = new List<double>();
            var firmEyesOpenCopSwayPca = new List<double>();
            var firmEyesOpenRmsX = new List<double>();
            var firmEyesOpenRmsY = new List<double>();
            var firmEyesClosedFiles = files.Where(m => m.Contains("Firm") && m.Contains("Eyes
Closed"));
            var firmEyesClosedCopVelocity = new List<double>();
            var firmEyesClosedCopswayPca = new List<double>();
            var firmEyesClosedRmsX = new List<double>();
            var firmEyesClosedRmsY = new List<double>();
            var firmDynamicFiles = files.Where(m => m.Contains("Firm") && m.Contains("Dynamic"));
            var firmDynamicCopVelocity = new List<double>();
            var firmDynamicCopswayPca = new List<double>();
            var firmDynamicRmsX = new List<double>();
            var firmDynamicRmsY = new List<double>();
            var foamEyesOpenFiles = files.Where(m => m.Contains("Foam") && m.Contains("Eyes
Open"));
            var foamEyesOpenCopVelocity = new List<double>();
            var foamEyesOpenCopSwayPca = new List<double>();
            var foamEyesOpenRmsX = new List<double>();
            var foamEyesOpenRmxY = new List<double>();
            var foamEyesClosedFiles = files.Where(m => m.Contains("Foam") && m.Contains("Eyes
Closed"));
            var foamEyesClosedCopVelocity = new List<double>();
            var foamEyesClosedCopSwayPca = new List<double>();
            var foamEyesClosedRmsX = new List<double>();
            var foamEyesClosedRmxY = new List<double>();
            var foamDynamicFiles = files.Where(m => m.Contains("Foam") && m.Contains("Dynamic"));
            var foamDynamicCopVelocity = new List<double>();
            var foamDynamicCopSwayPca = new List<double>();
            var foamDynamicRmsX = new List<double>();
            var foamDynamicRmxY = new List<double>();

            Parallel.ForEach(firmEyesOpenFiles, currentFile =>
```

```
            {
                using (var engine = new ExcelEngine())
                {
                    var application = engine.Excel;
                    var workbook = application.Workbooks.Open(currentFile);
                    var worksheet = workbook.Worksheets[0];
                    firmEyesOpenCopVelocity.Add(worksheet[3, 6].Number);
                    firmEyesOpenCopSwayPca.Add(worksheet[2, 6].Number);
                    firmEyesOpenRmsX.Add(worksheet[4, 6].Number);
                    firmEyesOpenRmsY.Add(worksheet[5, 6].Number);
                }
            });

            Parallel.ForEach(foamEyesOpenFiles, currentFile =>
            {
                using (var engine = new ExcelEngine())
                {
                    var application = engine.Excel;
                    var workbook = application.Workbooks.Open(currentFile);
                    var worksheet = workbook.Worksheets[0];
                    foamEyesOpenCopVelocity.Add(worksheet[3, 6].Number);
                    foamEyesOpenCopSwayPca.Add(worksheet[2, 6].Number);
                    foamEyesOpenRmsX.Add(worksheet[4, 6].Number);
                    foamEyesOpenRmxY.Add(worksheet[5, 6].Number);
                }
            });

            Parallel.ForEach(firmEyesClosedFiles, currentFile =>
            {
                using (var engine = new ExcelEngine())
                {
                    var application = engine.Excel;
                    var workbook = application.Workbooks.Open(currentFile);
                    var worksheet = workbook.Worksheets[0];
                    firmEyesClosedCopVelocity.Add(worksheet[3, 6].Number);
                    firmEyesClosedCopswayPca.Add(worksheet[2, 6].Number);
                    firmEyesClosedRmsX.Add(worksheet[4, 6].Number);
                    firmEyesClosedRmsY.Add(worksheet[5, 6].Number);
                }
            });

            Parallel.ForEach(foamEyesClosedFiles, currentFile =>
            {
                using (var engine = new ExcelEngine())
                {
                    var application = engine.Excel;
                    var workbook = application.Workbooks.Open(currentFile);
                    var worksheet = workbook.Worksheets[0];
                    foamEyesClosedCopVelocity.Add(worksheet[3, 6].Number);
                    foamEyesClosedCopSwayPca.Add(worksheet[2, 6].Number);
                    foamEyesClosedRmsX.Add(worksheet[4, 6].Number);
                    foamEyesClosedRmxY.Add(worksheet[5, 6].Number);
                }
            });

            Parallel.ForEach(firmDynamicFiles, currentFile =>
            {
                using (var engine = new ExcelEngine())
                {
                    var application = engine.Excel;
                    var workbook = application.Workbooks.Open(currentFile);
                    var worksheet = workbook.Worksheets[0];
                    firmDynamicCopVelocity.Add(worksheet[3, 6].Number);
                    firmDynamicCopswayPca.Add(worksheet[2, 6].Number);
                    firmDynamicRmsX.Add(worksheet[4, 6].Number);
                    firmDynamicRmsY.Add(worksheet[5, 6].Number);
                }
            });
```

```
            Parallel.ForEach(foamDynamicFiles, currentFile =>
            {
                using (var engine = new ExcelEngine())
                {
                    var application = engine.Excel;
                    var workbook = application.Workbooks.Open(currentFile);
                    var worksheet = workbook.Worksheets[0];
                    foamDynamicCopVelocity.Add(worksheet[3, 6].Number);
                    foamDynamicCopSwayPca.Add(worksheet[2, 6].Number);
                    foamDynamicRmsX.Add(worksheet[4, 6].Number);
                    foamDynamicRmxY.Add(worksheet[5, 6].Number);
                }
            });
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FirmEyesOpenResultsCopVelocity = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesOpenCopVelocity, 0)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesClosedCopVelocity, 0)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmDynamicCopVelocity, 0)
                    }
                };
            }).Wait();
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FirmEyesOpenRmsAp = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesOpenRmsY, 0)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesClosedRmsY, 0)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX = GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmDynamicRmsY,
0)
                    }
                };
            }).Wait();
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FirmEyesOpenRmsMl = new List<Result>
```

```
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesOpenRmsX, 0)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesClosedRmsX, 0)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX = GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmDynamicRmsX,
0)
                    }
                };
            });
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FoamEyesOpenRmsMl = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesOpenRmsX, 0)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesClosedRmsX, 0)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX = GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamDynamicRmsX,
0)
                    }
                };
            }).Wait();
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FoamEyesOpenRmsAp = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesOpenRmxY, 0)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesClosedRmxY, 0)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
```

```
                        PointX = GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamDynamicRmxY,
0)
                    }
                };
            }).Wait();
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FoamEyesOpenResultsCopVelocity = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesOpenCopVelocity, 0)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesClosedCopVelocity, 0)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamDynamicCopVelocity, 0)
                    }
                };
            });
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FirmEyesOpenResultsCopSway = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesOpenCopSwayPca, 0)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesClosedCopswayPca, 0)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmDynamicCopswayPca, 0)
                    }
                };
            }).Wait();
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FoamEyesOpenResultsCopSway = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesOpenCopSwayPca, 0)
                    },
```

```
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesClosedCopSwayPca, 0)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamDynamicCopSwayPca, 0)
                    }
                };
            }).Wait();
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FoamEyesOpenResultsCopSway = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesOpenCopSwayPca, 0)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesClosedCopSwayPca, 0)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamDynamicCopSwayPca, 0)
                    }
                };
            });
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FirmEyesClosedResultsCopSway = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesOpenCopSwayPca, 1)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesClosedCopswayPca, 1)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmDynamicCopswayPca, 1)
                    }
                };
            }).Wait();
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FirmEyesClosedResultsCopVelocity = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesOpenCopVelocity, 1)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesClosedCopVelocity, 1)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmDynamicCopVelocity, 1)
                    }
                };
            }).Wait();
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FirmEyesClosedRmsMl = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesOpenRmsX, 1)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesClosedRmsX, 1)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX = GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmDynamicRmsX,
1)
                    }
                };
            });
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FirmEyesClosedRmsAp = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesOpenRmsY, 1)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesClosedRmsY, 1)
```

```
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX = GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmDynamicRmsY,
1)
                    }
                };
            });
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FoamEyesClosedRmsAp = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesOpenRmxY, 1)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesClosedRmxY, 1)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX = GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamDynamicRmxY,
1)
                    }
                };
            }).Wait();
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FoamEyesClosedRmsMl = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesOpenRmsX, 1)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesClosedRmsX, 1)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX = GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamDynamicRmsX,
1)
                    }
                };
            }).Wait();
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FoamEyesClosedResultsCopVelocity = new List<Result>
                {
```

```
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesOpenCopVelocity, 1)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesClosedCopVelocity, 1)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamDynamicCopVelocity, 1)
                    }
                };
            });
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FoamEyesClosedResultsCopSway = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesOpenCopSwayPca, 1)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesClosedCopSwayPca, 1)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamDynamicCopSwayPca, 1)
                    }
                };
            }).Wait();
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FirmDynamicResultsCopSway = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesOpenCopSwayPca, 2)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesClosedCopswayPca, 2)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
```

```
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmDynamicCopswayPca, 2)
                    }
                };
            }).Wait();
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FoamDynamicResultsCopSway = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesOpenCopSwayPca, 2)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesClosedCopSwayPca, 2)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamDynamicCopSwayPca, 2)
                    }
                };
            });
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FirmDynamicResultsCopVelocity = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesOpenCopVelocity, 2)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesClosedCopVelocity, 2)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmDynamicCopVelocity, 2)
                    }
                };
            }).Wait();
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FirmDynamicAp = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
```

```
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesOpenRmsY, 2)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesClosedRmsY, 2)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX = GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmDynamicRmsY,
2)
                    }
                };
            });
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FoamDynamicRmsAp = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesOpenRmxY, 2)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesClosedRmxY, 2)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX = GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamDynamicRmxY,
2)
                    }
                };
            });
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FoamDynamicResultsCopVelocity = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesOpenCopVelocity, 2)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesClosedCopVelocity, 2)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamDynamicCopVelocity, 2)
                    }
                };
            }).Wait();
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FirmDynamicRmsMl = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesOpenRmsX, 2)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmEyesClosedRmsX, 2)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX = GetValueFromListAtIndexAndReturnZeroIfNotPresent(firmDynamicRmsX,
2)
                    }
                };
            }).Wait();
```

```
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (System.Action)
delegate
            {
                FoamDynamicRmsMl = new List<Result>
                {
                    new Result
                    {
                        CategoryType = "Eyes Open",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesOpenRmsX, 2)
                    },
                    new Result
                    {
                        CategoryType = "Eyes Closed",
                        PointX =
GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamEyesClosedRmsX, 2)
                    },
                    new Result
                    {
                        CategoryType = "Dynamic",
                        PointX = GetValueFromListAtIndexAndReturnZeroIfNotPresent(foamDynamicRmsX,
2)
                    }
                };
            }).Wait();
```

```
            CopVelocityMaxHeight = Math.Ceiling(GetMaxHeightForCopVelocity())*1.25;
            CopSwayPcaMaxHeight = Math.Ceiling(GetMaxHeightForCopSway())*1.25;
            RmsMlMaxHeight = Math.Ceiling(GetMaximHeightForRmsMl())*1.25;
            RmsApMaxHeight = Math.Ceiling(GetMaxHeightForAp())*1.25;
        }
```

```
        public double GetValueFromListAtIndexAndReturnZeroIfNotPresent(List<double> items, int
index)
        {
            try
            {
                return items[index];
            }
            catch (Exception)
            {
```

```
                return 0;
            }
        }

        public
            ICommand
            TreeViewLoaded
        {
            get { return _load; }
            set
            {
                _load = value;
                OnPropertyChanged("TreeViewLoaded");
            }
        }

        private
            bool CanLoad
            (object
                arg)
        {
            return true;
        }


        private
            List<Models.TreeModel> _parent;

        public
            TreeViewModel()
        {
            TreeViewLoaded = new DelegateCommand(LoadTree, CanLoad);
            LoadMostRecentPatient = new DelegateCommand(LoadPatient, CanLoad);
        }


        public
            List<Models.TreeModel> Parent
        {
            get { return _parent; }
            set
            {
                _parent = value;
                OnPropertyChanged("Parent");
            }
        }

        public ICommand LoadMostRecentPatient
        {
            get { return _loadMostRecent; }
            set
            {
                _loadMostRecent = value;
                OnPropertyChanged("LoadMostRecentPatient");
            }
        }

        private void LoadPatient(object obj)
        {
            var pathToItem = Patient.GetDirectoryNameOfMostRececnt();
            RenderSelectedItemValue(new Models.TreeModel
            {
                PathToItem = pathToItem
            });
        }

        public event
            PropertyChangedEventHandler PropertyChanged;
```

```
        private
            void LoadTree
            (object
                state)
        {
            var parentNode = new Models.TreeModel
            {
                PathToItem = SearchFolderPath,
                Header = "Results Explorer"
            };
            WalkTree(parentNode);
            Parent = new List<Models.TreeModel>
            {
                parentNode
            };
        }

        public static
            string SearchFolderPath =
Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) +
                                       @"/Vets/Results/";

        private
            List<Result> _firmEyesClosedResultsCopVelocity;

        private
            List<Result> _firmDynamicCopVelocity;

        private
            List<Result> _foamEyesOpenResultsCopVelocity;

        private
            List<Result> _foamEyesClosedResultsCopVelocity;

        private
            List<Result> _foamDynamicResultsCopVelocity;

        private
            List<Result> _firmEyesOpenCopSway;

        private
            List<Result> _firmEyesClosedResultsCopSway;

        private
            List<Result> _firmDynamicResultsCopSway;

        private
            List<Result> _foamEyesOpenResultsCopSway;

        private
            List<Result> _foamEyesClosedResultsCopSway;

        private
            List<Result> _foamDynamicResultsCopSway;

        private List<Result> _firmEyesOpenRmsMl;
        private List<Result> _firmEyesClosedRmsMl;
        private List<Result> _firmDynamicRmsMl;
        private List<Result> _foamEyesOpenRmsMl;
        private List<Result> _foamDynamicRmsMl;
        private List<Result> _foamEyesClosedRmsMl;
        private List<Result> _firmEyesOpenAp;
        private List<Result> _firmEyesClosedAp;
        private List<Result> _firmDynamicAp;
        private List<Result> _foamEyesOpenRmsAp;
        private List<Result> _foamEyesClosedRmsAp;
        private List<Result> _foamDynamicRmsAp;
        private double _copVelocityHeight;
        private double _copSwayPcaHeight;
```

```
        private double _rmsMlPcaHeight;
        private double _rmsApcaHeight;
        private ICommand _loadMostRecent;

        private
            void WalkTree
            (Models.TreeModel
                parent)
        {
            try
            {
                var directories = Directory.GetDirectories(parent.PathToItem);
                foreach (var model in directories.Select(directory => new Models.TreeModel
                {
                    Header = new DirectoryInfo(directory).Name,
                    PathToItem = directory
                }))
                {
                    parent.Models.Add(model);
                }

                foreach (var treeModel in parent.Models)
                {
                    WalkTree(treeModel);
                }
            }
            catch (Exception exception)
            {
                MessageBox.Show(exception.Message);
            }
        }

        [
            NotifyPropertyChangedInvocator]
        protected virtual
        void OnPropertyChanged
            ([
                CallerMemberName] string propertyName = null)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (handler != null) handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

RESULTS TREE MODEL

Used to provide container object for tree grid

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
// file:ViewModels\Views\Results\Models\TreeModel.cs
//
// summary:     Implements the tree model class
////////////////////////////////////////////////////////////////////////////////////////////////////
//

using System.Collections.ObjectModel;
using System.Collections.Specialized;
using System.ComponentModel;

namespace VetsLegacy.Presentation.Views.Results.Models
{

////////////////////////////////////////////////////////////////////////////////////////////////////
//
    /// <summary>   A data Model for the tree. </summary>
    /// <remarks>   Maxim, 5/11/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
```

```
    public class TreeModel : INotifyPropertyChanged
    {

////////////////////////////////////////////////////////////////////////////////////////////////////
//

        /// <summary>   Number of. </summary>
        private string _count;

        /// <summary>   The header. </summary>
        private string _header;

        /// <summary>   The icon. </summary>
        private string _icon;

        /// <summary>   The models. </summary>
        private ObservableCollection<TreeModel> _models;

        /// <summary>   The path to item. </summary>
        private string _pathToItem;

        public TreeModel()
        {
            Models = new ObservableCollection<TreeModel>();
            Models.CollectionChanged += Models_CollectionChanged;
        }

        public bool IsDirectory { get; set; }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the header. </summary>
        /// <value> The header. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public string Header
        {
            get { return _header; }
            set
            {
                _header = value;
                OnPropertyChanged("Header");
            }
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the models. </summary>
        /// <value> The models. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public ObservableCollection<TreeModel> Models
        {
            get { return _models; }
            set
            {
                _models = value;
                OnPropertyChanged("Models");
            }
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the path to item. </summary>
```

```
        /// <value> The path to item. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public string PathToItem
        {
            get { return _pathToItem; }
            set
            {
                _pathToItem = value;
                OnPropertyChanged("PathToItem");
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the number of. </summary>
        /// <value> The count. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public string Count
        {
            get { return _count; }
            set
            {
                _count = value;
                OnPropertyChanged("Count");
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the icon. </summary>
        /// <value> The icon. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public string Icon
        {
            get { return _icon; }
            set
            {
                _icon = value;
                OnPropertyChanged("Icon");
            }
        }

        /// <summary>   Event queue for all listeners interested in PropertyChanged events.
</summary>
        public event PropertyChangedEventHandler PropertyChanged;

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Event handler. Called by Models for collection changed events. </summary>
        /// <remarks>   Maxim, 5/15/2014. </remarks>
        /// <param name="sender">   Source of the event. </param>
        /// <param name="e">        Notify collection changed event information. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void Models_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)
        {
            if (Models.Count > 0)
                Count = "( " + Models.Count + " )";
```

```
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Executes the property changed action. </summary>
        /// <remarks>   Maxim, 5/11/2014. </remarks>
        /// <param name="name"> The name. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public void OnPropertyChanged(string name)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs(name));
        }
    }
}
```

PLAY VIEW

Used to provide user option to review results before starting trials

```
<UserControl xmlns:syncfusion="http://schemas.syncfusion.com/wpf"
x:Class="VetsLegacy.Presentation.Views.Play.Play"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
             xmlns:play1="clr-namespace:VetsLegacy.Presentation.Views.Play"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
    <UserControl.DataContext>
        <play1:PlayViewModel />
    </UserControl.DataContext>
    <i:Interaction.Triggers>
        <i:EventTrigger EventName="Loaded">
            <i:InvokeCommandAction Command="{Binding WindowLoaded}" />
        </i:EventTrigger>
    </i:Interaction.Triggers>
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width=".5*" />
        </Grid.ColumnDefinitions>
        <Grid Grid.Column="0">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="*" />
            </Grid.RowDefinitions>
            <Label Content="Trial List For Current User" FontWeight="Bold" />
            <ScrollViewer Grid.Row="1">
                <syncfusion:GridDataControl
                    x:Name="TrialGridDataControl"
                    AllowEdit="False"
                    AllowSelection="None"
                    AutoPopulateColumns="False"
                    AutoPopulateRelations="False"
                    ColumnSizer="Star"
                    ContextMenuOptions="Default"
                    EnableContextMenu="True"
                    IsDynamicItemsSource="True"
                    ItemsSource="{Binding Trials}"
                    NotifyPropertyChanges="True"
                    ShowAddNewRow="False"
                    VisualStyle="Metro"
                    ListBoxSelectionMode="MultiExtended">
                    <syncfusion:GridDataControl.VisibleColumns>
                        <syncfusion:GridDataVisibleColumn MappingName="SessionName">
                            <syncfusion:GridDataVisibleColumn.HeaderStyle>
```

```
                            <syncfusion:GridDataColumnStyle HorizontalAlignment="Center" />
                        </syncfusion:GridDataVisibleColumn.HeaderStyle>
                        <syncfusion:GridDataVisibleColumn.ColumnStyle>
                            <syncfusion:GridDataColumnStyle HorizontalAlignment="Center"
Foreground="Black" />
                        </syncfusion:GridDataVisibleColumn.ColumnStyle>
                    </syncfusion:GridDataVisibleColumn>
                    <syncfusion:GridDataVisibleColumn MappingName="TrialOption">
                        <syncfusion:GridDataVisibleColumn.HeaderStyle>
                            <syncfusion:GridDataColumnStyle HorizontalAlignment="Center" />
                        </syncfusion:GridDataVisibleColumn.HeaderStyle>
                        <syncfusion:GridDataVisibleColumn.ColumnStyle>
                            <syncfusion:GridDataColumnStyle HorizontalAlignment="Center"
Foreground="Black" />
                        </syncfusion:GridDataVisibleColumn.ColumnStyle>
                    </syncfusion:GridDataVisibleColumn>
                    <syncfusion:GridDataVisibleColumn MappingName="Priority">
                        <syncfusion:GridDataVisibleColumn.HeaderStyle>
                            <syncfusion:GridDataColumnStyle HorizontalAlignment="Center" />
                        </syncfusion:GridDataVisibleColumn.HeaderStyle>
                        <syncfusion:GridDataVisibleColumn.ColumnStyle>
                            <syncfusion:GridDataColumnStyle HorizontalAlignment="Center"
Foreground="Black" />
                        </syncfusion:GridDataVisibleColumn.ColumnStyle>
                    </syncfusion:GridDataVisibleColumn>
                </syncfusion:GridDataControl.VisibleColumns>
            </syncfusion:GridDataControl>
        </ScrollViewer>
    </Grid>
    <Grid Grid.Column="1" Margin="5">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Grid Margin="2" Grid.Row="0">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="*" />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
            <StackPanel Orientation="Vertical" Grid.Row="1" Grid.Column="1">
                <TextBlock Text="Subject Info" FontWeight="Bold"
                       HorizontalAlignment="Center" />
                <syncfusion:SfTextBoxExt AutoCompleteSource="{Binding SuggestionsList}"
                                         AutoCompleteMode="SuggestAppend"
                                         SearchItemPath="Name" SuggestionMode="Contains"
x:Name="PatientNameTextBox">
                    <syncfusion:SfTextBoxExt.AutoCompleteItemTemplate>
                        <DataTemplate>
                            <TextBlock Text="{Binding Name}" />
                        </DataTemplate>
                    </syncfusion:SfTextBoxExt.AutoCompleteItemTemplate>
                </syncfusion:SfTextBoxExt>
                <TextBlock Text="Session Storage Name" FontWeight="Bold"
                           HorizontalAlignment="Center"/>
                <TextBox Text="{Binding SessionName}" FontWeight="Normal"
HorizontalAlignment="Stretch"/>
            </StackPanel>
        </Grid>
        <Grid x:Name="VerifySettingsBox" Grid.Row="1" VerticalAlignment="Center">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="*" />
            </Grid.RowDefinitions>
```

```
                <TextBlock Text="Verify Settings Below" FontWeight="Bold"
HorizontalAlignment="Center" Grid.Row="0" />
                <Grid Grid.Row="1" Margin="2">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="*"/>
                        <RowDefinition Height="*"/>
                        <RowDefinition Height="*"/>
                    </Grid.RowDefinitions>
                    <TextBlock Text="Sample Rate" FontWeight="Bold" Foreground="Black"
VerticalAlignment="Center" HorizontalAlignment="Center" Grid.Row="0" Grid.Column="0"/>
                    <TextBox Text="{Binding SampleRate,Mode=TwoWay}" Grid.Column="1"
Grid.Row="0"/>
                    <TextBlock Text="Wait Time" FontWeight="Bold" Foreground="Black"
VerticalAlignment="Center" HorizontalAlignment="Center" Grid.Row="1" Grid.Column="0"/>
                    <TextBox Text="{Binding WaitTime,Mode=TwoWay}" Grid.Column="1" Grid.Row="1"/>
                    <TextBlock Text="Run Time" FontWeight="Bold" Foreground="Black"
VerticalAlignment="Center" HorizontalAlignment="Center" Grid.Row="2" Grid.Column="0" />
                    <TextBox Text="{Binding RunTime,Mode=TwoWay}" Grid.Column="1" Grid.Row="2"/>
                </Grid>
            </Grid>
            <Grid x:Name="StartButtonGrid" Grid.Row="2" Margin="2" VerticalAlignment="Center">
                <Button Content="Start" Command="{Binding Play}" CommandParameter="{Binding
ElementName=PatientNameTextBox,Path=Text}"/>
            </Grid>
        </Grid>
    </Grid>
</UserControl>
```

PLAY BACK END LOGIC

Used to provide back end logic for play

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Diagnostics;
using System.IO;
using System.Runtime.CompilerServices;
using System.Windows.Input;
using Syncfusion.Windows.Shared;
using VetsLegacy.Models.User;
using VetsLegacy.Models.User.Trial;
using VetsLegacy.Presentation.Simulation;
using VetsLegacy.Presentation.Views.Play.AutoComplete;

namespace VetsLegacy.Presentation.Views.Play
{
    public class PlayViewModel : INotifyPropertyChanged
    {
        private Settings.Settings _currentSetting;
        private ICommand _dropDownKeyPressed;
        private ICommand _loaded;
        private ICommand _play;
        private double _runTime;

        private double _sampleRate;
        private string _sessionName;
        private List<SuggestionItem> _suggestionsList;
        private List<Trial> _trials;
        private double _waitTime;

        public PlayViewModel()
        {
            WindowLoaded = new DelegateCommand(InitializeTrialsToList, o => true);
            Play = new DelegateCommand(PlayAnimation, o => true);
            LoadSuggestionsList();
```

```
            SetCurrentSetting();
        }

        public double SampleRate
        {
            get { return _sampleRate; }
            set
            {
                _sampleRate = value;
                OnPropertyChanged("SampleRate");
            }
        }

        public double WaitTime
        {
            get { return _waitTime; }
            set
            {
                _waitTime = value;
                OnPropertyChanged("WaitTime");
            }
        }

        public string SessionName
        {
            get { return _sessionName; }
            set
            {
                _sessionName = value;
                OnPropertyChanged("SessionName");
            }
        }

        public double RunTime
        {
            get { return _runTime; }
            set
            {
                _runTime = value;
                OnPropertyChanged("RunTime");
            }
        }

        public ICommand Play
        {
            get { return _play; }
            set
            {
                _play = value;
                OnPropertyChanged("Play");
            }
        }

        public List<Trial> Trials
        {
            get { return _trials; }
            set
            {
                _trials = value;
                OnPropertyChanged("Trials");
            }
        }

        public Settings.Settings CurrentSetting
        {
            get { return _currentSetting; }
            set
            {
                _currentSetting = value;
```

```
                OnPropertyChanged("CurrentSetting");
            }
        }

        public ICommand WindowLoaded
        {
            get { return _loaded; }
            set
            {
                _loaded = value;
                OnPropertyChanged("WindowLoaded");
            }
        }

        public List<SuggestionItem> SuggestionsList
        {
            get { return _suggestionsList; }
            set
            {
                _suggestionsList = value;
                OnPropertyChanged("SuggestionsList");
            }
        }

        public ICommand DropDownKeyPressed
        {
            get { return _dropDownKeyPressed; }
            set
            {
                _dropDownKeyPressed = value;
                OnPropertyChanged("DropDownKeyPressed");
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;

        private void PlayAnimation(object obj)
        {
            SetCurrentSettingToValues();
            AddPatient(obj);
            RunAnimation();
        }

        private void RunAnimation()
        {
            AnimateSimulationViewModel animateSimulationViewModel = SessionName != String.Empty ?
new AnimateSimulationViewModel(SessionName) : new AnimateSimulationViewModel();
            var animation = new AnimateSimulationWindow { DataContext = animateSimulationViewModel
};
            animation.ShowDialog();
        }

        private void AddPatient(object obj)
        {
            var patient = (string) obj;
            if (Patient.IsValidPatient(patient))
            {
                Patient.DeletePatient(Patient.GetPatientFromString(patient));
                Patient.AddPatient(patient);
            }
            Patient.AddPatient(patient);
        }

        private void SetCurrentSettingToValues()
        {
            Models.User.Settings.Settings.SetCurrentSettingAsDefault(new
Models.User.Settings.Settings
            {
                SamplingRate = SampleRate,
```

```
                TimePerTrial = RunTime,
                WaitTimeBetweenSessions = WaitTime
            });
        }

        private void InitializeTrialsToList(object parameter = null)
        {
            Trials = Trial.Trials;
        }

        private void LoadSuggestionsList()
        {
            string path = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) +
@"/Vets/Results/";
            IEnumerable<string> patientNames = Directory.EnumerateDirectories(path, "*",
SearchOption.TopDirectoryOnly);
            SuggestionsList = new List<SuggestionItem>();
            foreach (string name in patientNames)
            {
                SuggestionsList.Add(new SuggestionItem
                {
                    Name = new DirectoryInfo(name).Name
                });
            }
        }

        private void SetCurrentSetting()
        {
            var settings = Models.User.Settings.Settings.GetCurrentSettings();
            if (settings == null)
            {
                RunTime = 60;
                WaitTime = 30;
                SampleRate = 100;
            }
            else
            {
                RunTime = settings.TimePerTrial;
                WaitTime = settings.WaitTimeBetweenSessions;
                SampleRate = settings.SamplingRate;
            }
        }
        protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (handler != null) handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

ANIMATION

Used to provide animation system for actual trials

```
<Window x:Class="VetsLegacy.Presentation.Simulation.AnimateSimulationWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
        xmlns:simulation1="clr-namespace:VetsLegacy.Presentation.Simulation" Name="Window"
        WindowState="Maximized" WindowStyle="None" WindowStartupLocation="CenterScreen"
        KeyDown="AnimateSimulationWindow_OnKeyDown" Unloaded="AnimateSimulationWindow_OnUnloaded">
    <Window.DataContext>
        <simulation1:AnimateSimulationViewModel />
    </Window.DataContext>
    <i:Interaction.Triggers>
        <i:EventTrigger EventName="KeyDown">
            <i:InvokeCommandAction Command="{Binding KeyPressed}" />
        </i:EventTrigger>
        <i:EventTrigger EventName="MouseDown">
            <i:InvokeCommandAction Command="{Binding KeyPressed}"/>
        </i:EventTrigger>
```

```
    </i:Interaction.Triggers>
    <Window.Resources>
        <Storyboard x:Key="RotateStoryboard">
            <DoubleAnimation x:Name="RotateImageAnimation"
                             Storyboard.TargetName="Image"
                             Storyboard.TargetProperty="RenderTransform.Angle"
                             From="0"
                             To="{Binding Rotation}"
                             Duration="{Binding Duration}"
                             AutoReverse="False"
                             RepeatBehavior="Forever"
                             DesiredFrameRate="60"
                             />
        </Storyboard>
    </Window.Resources>
    <Grid Background="Black" x:Name="Grid" RenderTransformOrigin="0.5 0.5">
        <Grid.RenderTransform>
            <ScaleTransform ScaleX="1.5" ScaleY="1.5" />
        </Grid.RenderTransform>
        <TextBlock Text="{Binding WaitMessage}" HorizontalAlignment="Center"
VerticalAlignment="Center"
                   Foreground="White" FontWeight="Bold" />
        <Image Source="{Binding InputImage}" Visibility="{Binding ImageVisible}" x:Name="Image"
               RenderTransformOrigin=".5 .43"
               RenderOptions.BitmapScalingMode="LowQuality">
            <Image.RenderTransform>
                <RotateTransform CenterX=".5" CenterY=".5" Angle="0" />
            </Image.RenderTransform>
        </Image>
    </Grid>
</Window>
```

ANIMATION BACK END LOGIC

Used to provide back end logic for animation system

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// file:ViewModels\Simulation\AnimateSimulationViewModel.cs
//
// summary:     Implements the animate simulation view model class
////////////////////////////////////////////////////////////////////////////////////////////////
//

using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Timers;
using System.Windows;
using System.Windows.Input;
using System.Windows.Interop;
using System.Windows.Media.Animation;
using System.Windows.Media.Imaging;
using System.Windows.Threading;
using Syncfusion.Windows.Shared;
using Syncfusion.XlsIO;
using VetsLegacy.Extensions;
using VetsLegacy.Models.ExternalWrappers.BalanceBoard;
using VetsLegacy.Models.Timer;
using VetsLegacy.Models.User;
using VetsLegacy.Models.User.Settings;
using VetsLegacy.Models.User.Trial;
using VetsLegacy.Models.Writers;
using VetsLegacy.Presentation.Views.MainWindow;
using VetsLegacy.Presentation.Views.Results;
using VetsLegacy.Presentation.Views.Results.ResultsPatientTree;
```

```
namespace VetsLegacy.Presentation.Simulation
{

////////////////////////////////////////////////////////////////////////////////////////////////////
//
    /// <summary>   A ViewModel for the animate simulation. </summary>
    /// <remarks>   Maxim, 4/29/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
    public class AnimateSimulationViewModel : INotifyPropertyChanged
    {
        private readonly string _sessionName;

        /// <summary>   The board. </summary>
        private BalanceBoard _board;

        /// <summary>   The duration. </summary>
        private double _duration;

        /// <summary>   The image rotation. </summary>
        private Storyboard _imageRotation;

        /// <summary>   The image visible. </summary>
        private Visibility _imageVisible;

        /// <summary>   Renders the full. </summary>
        /// <remarks>   Maxim, 4/29/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private bool _isClockwise = true;

        /// <summary>   true if this object is key pressed. </summary>
        private bool _isKeyPressed;

        /// <summary>   The key pressed. </summary>
        private ICommand _keyPressed;

        private MicroTimer _microTimer;
        private double _rotation;

        /// <summary>   Options for controlling the operation. </summary>
        private Settings _settings;

        /// <summary>   Name of the trial patient. </summary>
        private string _trialPatientName;

        /// <summary>   The trials. </summary>
        private ConcurrentQueue<Trial> _trials;

        /// <summary>   Message describing the wait. </summary>
        private string _waitMessage;

        private BitmapImage _image;


////////////////////////////////////////////////////////////////////////////////////////////////////
//


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Default constructor. </summary>
        /// <remarks>   Maxim, 4/29/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public AnimateSimulationViewModel()
```

```
        {
            _sessionName = GetNewName();
            Init();
        }

        public AnimateSimulationViewModel(string name)
        {
            _sessionName = name;
            Init();
        }

        public ICommand KeyPressed
        {
            get { return _keyPressed; }
            set
            {
                _keyPressed = value;
                OnPropertyChanged("KeyPressed");
            }
        }

        public double Duration
        {
            get { return _duration; }
            set
            {
                _duration = value;
                OnPropertyChanged("Duration");
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets a message describing the wait. </summary>
        /// <value> A message describing the wait. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public string WaitMessage
        {
            get { return _waitMessage; }
            set
            {
                _waitMessage = value;
                OnPropertyChanged("WaitMessage");
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the image visible. </summary>
        /// <value> The image visible. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public Visibility ImageVisible
        {
            get { return _imageVisible; }
            set
            {
                _imageVisible = value;
                OnPropertyChanged("ImageVisible");
            }
        }
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Loads the animation. </summary>
        /// <remarks>   Maxim, 4/29/2014. </remarks>
        public double Rotation
        {
            get { return _rotation; }
            set
            {
                _rotation = value;
                OnPropertyChanged("Rotation");
            }
        }

        /// <summary>   Event queue for all listeners interested in PropertyChanged events.
</summary>
        public event PropertyChangedEventHandler PropertyChanged;

        private string GetNewName()
        {
            return "Session";
        }

        private void Init()
        {
            InitialSimulationSettings();
            ConnectToBalanceBoard();
            ConnectToBalanceBoard();
            RenderFull();
            KeyPressed = new DelegateCommand(KeyDownEvent, CanKeyDown);
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Determine if we can key down. </summary>
        /// <remarks>   Maxim, 5/15/2014. </remarks>
        /// <param name="arg">  The argument. </param>
        /// <returns>   true if we can key down, false if not. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private bool CanKeyDown(object arg)
        {
            return true;
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Key down event. </summary>
        /// <remarks>   Maxim, 4/29/2014. </remarks>
        /// <param name="obj">  The object. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void KeyDownEvent(object obj)
        {
            _isKeyPressed = true;
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//

        public BitmapImage InputImage
        {
```

```
            get { return new BitmapImage(new Uri("/Images/WiiModuleStaticImage.PNG",
UriKind.RelativeOrAbsolute)); }
        }

        private void RenderFull()
        {
            RenderWaitScreen();
        }

////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Renders the wait screen. </summary>
        /// <remarks>   Maxim, 4/29/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void RenderWaitScreen()
        {
            Trial checkTrial;
            if (_trials.TryPeek(out checkTrial))
            {
                if (checkTrial != null)
                {
                    KeyPressed = null;
                    ImageVisible = Visibility.Collapsed;
                    double waitTime = GetCurrentWaitTime();
                    var timer = new Timer(1000);
                    timer.Elapsed += delegate
                    {
                        if (Math.Abs(waitTime) < 0.01)
                        {
                            timer.Stop();
                            WaitMessage = checkTrial.TrialOption == TrialOption.Firm
                                ? "Take Off Foam And "
                                : "Add Foam And ";
                            switch (checkTrial.SessionType)
                            {
                                case TrialType.Black:
                                    WaitMessage += "Close Eyes";
                                    break;
                                default:
                                    WaitMessage += "Open Eyes";
                                    break;
                            }
                            WaitMessage += " And Press Any Key To Continue";
                            SetKeyPressToNewCommand(RenderNext);
                        }
                        else
                        {
                            WaitMessage = waitTime + " Seconds To Start";
                            waitTime--;
                        }
                    };
                    timer.Start();
                }
            }
            else
            {
                RenderExitScreen();
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Sets key press to new command. </summary>
        /// <remarks>   Maxim, 5/15/2014. </remarks>
        /// <param name="renderNext">   The render next. </param>
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void SetKeyPressToNewCommand(Action renderNext)
        {
            KeyPressed = new DelegateCommand(o => renderNext(), o => true);
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets current wait time. </summary>
        /// <remarks>   Maxim, 5/15/2014. </remarks>
        /// <returns>   The current wait time. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private int GetCurrentWaitTime()
        {
            return (int) _settings.WaitTimeBetweenSessions;
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Renders the exit screen. </summary>
        /// <remarks>   Maxim, 4/29/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void RenderExitScreen()
        {
            ImageVisible = Visibility.Collapsed;
            WaitMessage = "Please Press Any Key To Exit";
            KeyPressed = new DelegateCommand(Close, o => true);
        }

        private void Close(object state)
        {
            var window = Application.Current.Windows.OfType<Window>().FirstOrDefault(x =>
x.IsActive);
            window.Close();
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Renders the next. </summary>
        /// <remarks>   Maxim, 4/29/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void RenderNext()
        {
            StopRotationIfActive();
            DisactivateKeyPress();
            Trial currentTrial = GetNextTrial();
            if (currentTrial == null) return;
            switch (currentTrial.SessionType)
            {
                case TrialType.Black:
                    ImageVisible = Visibility.Hidden;
                    WaitMessage = String.Empty;
                    Record(currentTrial.SessionType, currentTrial.SessionName,
currentTrial.TrialOption, currentTrial);
                    break;
                case TrialType.Static:
                    ImageVisible = Visibility.Visible;
                    WaitMessage = String.Empty;
```

```
                    Record(currentTrial.SessionType, currentTrial.SessionName,
currentTrial.TrialOption, currentTrial);
                    break;
                case TrialType.Dynamic:
                    ImageVisible = Visibility.Visible;
                    WaitMessage = String.Empty;
                    RunAnimation();
                    Record(currentTrial.SessionType, currentTrial.SessionName,
currentTrial.TrialOption, currentTrial);
                    break;
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets the next trial. </summary>
        /// <remarks>   Maxim, 5/20/2014. </remarks>
        /// <returns>   The next trial. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////
//
        private Trial GetNextTrial()
        {
            Trial currentTrial;
            _trials.TryDequeue(out currentTrial);
            return currentTrial;
        }

////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Disactivate key press. </summary>
        /// <remarks>   Maxim, 5/15/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void DisactivateKeyPress()
        {
            KeyPressed = null;
        }

////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Stops rotation if active. </summary>
        /// <remarks>   Maxim, 5/15/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void StopRotationIfActive()
        {
            try
            {
                _imageRotation.Stop();
            }
            catch (Exception)
            {
            }
        }


////////////////////////////////////////////////////////////////////////////////////////////////
//

        private void RunAnimation()
        {
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (Action) delegate
```

```
            {
                if (_isClockwise)
                {
                    Rotation = 360;
                }
                else
                {
                    Rotation = -360;
                }
                _isClockwise = !_isClockwise;
                Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (Action)
delegate
                {
                    Window Window =
                        Application.Current.Windows.OfType<Window>().FirstOrDefault(x =>
x.IsActive);
                    _imageRotation = (Storyboard) Window.TryFindResource("RotateStoryboard");
                    _imageRotation.SpeedRatio = .20;
                    _imageRotation.Begin();
                });
            });
        }

        private int _counter;

        private void Record(TrialType sessionType, string name, TrialOption option, Trial
currentTrial)
        {
            Dispatcher.CurrentDispatcher.BeginInvoke(DispatcherPriority.Render, (Action) delegate
            {
                var points = new List<BalanceBoardPoint>();
                _counter = 0;
                double totalIterations = _settings.SamplingRate*_settings.TimePerTrial;
                double sampleRate = 1000/_settings.SamplingRate;

                _microTimer = new MicroTimer((long) sampleRate*1000);
                _microTimer.MicroTimerElapsed += delegate
                {
                    if (Math.Abs(_counter - totalIterations) < 0.01)
                    {
                            ImageVisible = Visibility.Hidden;
                            WaitMessage = "Writing Data To File";
                            _microTimer.Stop();
                            Patient patient = Patient.GetLastAddedPatient();
                            CombineTrialAndPatienAndStore(currentTrial, points, patient);
                            WriteToFile(patient, currentTrial, option);
                            points = new List<BalanceBoardPoint>();
                            RenderWaitScreen();
                    }
                    else
                    {
                        Dispatcher.CurrentDispatcher.BeginInvoke((Action) delegate
                        {
                            points.Add(GetPoint());
                            _counter++;
                        }).Wait();
                    }
                };
                _microTimer.Start();
            });
        }

        public void AddToList(List<Trial> returnList, BalanceBoardPoint point)
        {

        }

////////////////////////////////////////////////////////////////////////////////////////////
//
```

```
        /// <summary>   Combine trial and patien and store. </summary>
        /// <remarks>   Maxim, 5/11/2014. </remarks>
        /// <param name="trial">    The trial. </param>
        /// <param name="points">   The points. </param>
        /// <param name="patient">  The patient. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void CombineTrialAndPatienAndStore(Trial trial, List<BalanceBoardPoint> points,
Patient patient)
        {
            trial.AddPointsToList(points);
            patient.AddTrial(trial);
            Patient.AddPatient(patient);
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Creates and set if no exists. </summary>
        /// <remarks>   Maxim, 5/7/2014. </remarks>
        /// <param name="fileName"> Filename of the file. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void CreateAndSetIfNoExists(string fileName)
        {
            if (!Directory.Exists(fileName) && fileName != null)
            {
                Directory.CreateDirectory(fileName);
            }
            if (fileName != null) Directory.SetCurrentDirectory(fileName);
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets directory path with patient name. </summary>
        /// <remarks>   Maxim, 5/14/2014. </remarks>
        /// <param name="name"> The name. </param>
        /// <returns>   The directory path with patient name. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private string GetDirectoryPathWithPatientName(string name)
        {
            return Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) +
@"/Vets/Results/" + name;
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Writes to file. </summary>
        /// <remarks>   Maxim, 5/7/2014. </remarks>
        /// <param name="patient">  The patient. </param>
        /// <param name="trial">    The trial. </param>
        /// <param name="option">   The option. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void WriteToFile(Patient patient, Trial trial, TrialOption option)
        {
            string directory = CreatedirectoryTreeForPatientResults(
                GetDirectoryPathWithPatientName(patient.PatientName),
                option,
                trial.SessionName);
```

```
            string fileName = ISO_Date() + Abbrevations.Convert(trial.SessionName) + "_" +
trial.TrialOption + "_" +
                                patient.PatientName + ".xlsx";
            Patient.SetDirectoryNameOfMostRecent(directory);
            var excel = new ExcelEngine();
            IApplication application = excel.Excel;
            IWorkbook workbook = application.Workbooks.Create(1);
            IWorksheet worksheet = workbook.Worksheets[0];
            int counter = 2;
            SetColumnTitlesForData(worksheet);
            foreach (BalanceBoardPoint point in trial.Points)
            {
                worksheet.SetNumber(counter, 1, point.CogX);
                worksheet.SetNumber(counter, 2, point.CogY);
                counter++;
            }
            WriteAnalysisValuesToFile(worksheet, GetPointsFromTrial(trial));
            SetWorkbookAuthorAndVersion(workbook);
            SaveAndDisposeWorkBook(workbook, fileName, excel);
        }

        public static String ISO_Date()
        {
            return DateTime.Now.ToString("MMddyyyy_HHmmss_");
        }

        private string CreatedirectoryTreeForPatientResults(string directoryPathWithPatientName,
TrialOption option,
            string trialName)
        {
            CreateAndSetIfNoExists(directoryPathWithPatientName);
            CreateAndSetIfNoExists(DateTime.Now.ToLongDateString());
            CreateAndSetIfNoExists(_sessionName);
            CreateAndSetIfNoExists(option.ToString());
            CreateAndSetIfNoExists(trialName);

            return directoryPathWithPatientName + @"\" + DateTime.Now.ToLongDateString() + @"\" +
_sessionName + @"\" +
                   option + @"\" + trialName;
        }


////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Writes the analysis values to file. </summary>
        /// <remarks>   Maxim, 5/14/2014. </remarks>
        /// <param name="worksheet">    The worksheet. </param>
        /// <param name="points">       The points. </param>

////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void WriteAnalysisValuesToFile(IWorksheet worksheet, List<BalanceBoardPoint>
points)
        {
            using (var writer = new ExcelWriter())
            {
                writer.WriteCopVelocityToFile(worksheet, points, _settings.SamplingRate,
_settings.TimePerTrial);
                writer.WriteCopSwayToFile(worksheet, points);
                writer.WriteRmsToFile(worksheet, points);
            }
        }

        private void SaveAndDisposeWorkBook(IWorkbook workbook, string name, ExcelEngine excel)
        {
            workbook.Version = ExcelVersion.Excel2013;
            workbook.SaveAs(name);
            workbook.Close();
            excel.Dispose();
```

```
        }

        ////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        /// <summary>   Gets points from trial. </summary>
        /// <remarks>   Maxim, 5/14/2014. </remarks>
        /// <param name="trial">    The trial. </param>
        /// <returns>   The points from trial. </returns>

        ////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        private List<BalanceBoardPoint> GetPointsFromTrial(Trial trial)
        {
            return trial.Points.ToList();
        }


        ////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        /// <summary>   Sets column titles for data. </summary>
        /// <remarks>   Maxim, 5/14/2014. </remarks>
        /// <param name="worksheet">    The worksheet. </param>

        ////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        private void SetColumnTitlesForData(IWorksheet worksheet)
        {
            worksheet.SetValue(1, 1, "Cog X");
            worksheet.SetValue(1, 2, "Cog Y");
            worksheet.SetValue(1, 3, "Time Stamp (MS)");
        }


        ////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        /// <summary>   Sets workbook author and version. </summary>
        /// <remarks>   Maxim, 5/14/2014. </remarks>
        /// <param name="workbook"> The workbook. </param>

        ////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        private void SetWorkbookAuthorAndVersion(IWorkbook workbook)
        {
            workbook.Author = "Virtual Environment TBI Screening";
            workbook.Version = ExcelVersion.Excel2013;
        }


        ////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        /// <summary>   Gets the point. </summary>
        /// <remarks>   Maxim, 4/29/2014. </remarks>
        /// <returns>   The point. </returns>

        ////////////////////////////////////////////////////////////////////////////////////////////////////
        //
        private BalanceBoardPoint GetPoint()
        {
            if (_board.IsConnected)
            {
                return _board.GetPoint();
            }
            return new BalanceBoardPoint
            {
                CogX = 0,
                CogY = 0,
                BottomLeft = 0,
                BottomRight = 0,
```

```
                TopLeft = 0,
                TopRight = 0
            };
        }

////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Initial simulation settings. </summary>
        /// <remarks>   Maxim, 4/29/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void InitialSimulationSettings()
        {
            _settings = Settings.GetCurrentSettings();
            _duration = _settings.TimePerTrial;
            _trialPatientName = Patient.GetLastAddedPatient().PatientName;
            _trials = Trial.ReturnTrialAsQueue();
        }

////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Connects to balance board. </summary>
        /// <remarks>   Maxim, 4/29/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////
//
        private void ConnectToBalanceBoard()
        {
            try
            {
                BalanceBoard.DisposeCurrent();
                _board = new BalanceBoard();
                _board.Connect();
            }
            catch
            {
                // do nothing, run in demo mode
            }
        }

        protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (handler != null) handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

CLASS CALCULATIONS

Used to provide calculation for common values(COP, RMS)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using Accord.Statistics.Analysis;
using AForge;
using MIConvexHull;
using VetsInternalNonLegacy.Domain.Analysis;
using VetsLegacy.Models.ExternalWrappers.BalanceBoard;

namespace VetsLegacy.Models.Analysis
{
    public class ClassCalculations
    {
        public double GetCopSwayAreaPca(double[,] data)
        {
```

```
            var dataX = new List<double>();
            var dataY = new List<double>();

            for (int i = 0; i < data.Length/2; i++)
            {
                dataX.Add(data[i, 0]);
                dataY.Add(data[i, 1]);
            }
            var meanX = dataX.Average();
            var meanY = dataY.Average();

            for (int i = 0; i < data.Length/2; i++)
            {
                data[i, 0] = dataX[i] - meanX;
                data[i, 1] = dataY[i] - meanY;
            }

            PrincipalComponentAnalysis pca = new PrincipalComponentAnalysis(data);
            pca.Compute();
            double[,] components = pca.Transform(data);

            var componentX = new List<double>();
            var componentY = new List<double>();

            for (int i = 0; i < (components.Length/2); i++)
            {
                componentX.Add(components[i, 0]);
                componentY.Add(components[i, 1]);
            }
            var maxX = componentX.Max();
            var maxY = componentY.Max();
            var minX = componentX.Min();
            var minY = componentY.Min();

            return Math.PI*(maxX - minX)/2*(maxY - minY)/2;
        }

        public double GetCopSwayAreaConvex(double[,] data)
        {
            var N = data.Length/2;
            var vertices = new Vertex[N];
            for (var i = 0; i < N; i++)
                vertices[i] = new Vertex(data[i, 0], data[i, 1]);

            var delanay = Triangulation.CreateDelaunay(vertices).Cells.ToList();


            var triangulation = (from cell in delanay
                let determinantA =
                    (cell.Vertices[0].Position[0] - cell.Vertices[2].Position[0])*
                    (cell.Vertices[1].Position[1] - cell.Vertices[2].Position[1])
                let determinantB =
                    (cell.Vertices[0].Position[1] - cell.Vertices[2].Position[1])*
                    (cell.Vertices[1].Position[0] - cell.Vertices[2].Position[0])
                select (double) 1/2*Math.Abs((determinantA - determinantB))).ToList();

            return triangulation.Sum();
        }

        public double GetRms(double[] data)
        {
            var N = data.Length;
            var dataSquare = new double[N];
            for (int i = 0; i < N; i++)
            {
                dataSquare[i] = data[i]*data[i];
            }
            return Math.Sqrt(dataSquare.Sum()/data.Length);
        }
```

```
        public double GetStd(double[] data)
        {
            var N = data.Length;
            var average = new double[N];
            var value = data.Average();
            for (int i = 0; i < N; i++)
            {
                average[i] = Math.Pow((data[i] - value), 2);
            }
            return Math.Sqrt(average.Sum()/N);
        }

        public double GetCopVelocity(double[] data)
        {
            var N = data.Length;
            var dataPrime = new double[N - 1];
            for (int i = 0; i < N - 1; i++)
            {
                dataPrime[i] = data[i + 1] - data[i];
            }
            var velocity = dataPrime.Average();
            return velocity;
        }

        public double GetCopVelocitySampleRate(List<BalanceBoardPoint> points,double
sampleRate,double time)
        {
            var results = new List<double>();
            double[] dataX = new double[points.Count() - 1];
            double[] dataY = new double[points.Count() - 1];
            for (int counter = 0; counter < points.Count()-1; counter++)
            {
                dataX[counter] = Math.Pow(points[counter + 1].CogX - points[counter].CogX, 2);
                dataY[counter] = Math.Pow(points[counter + 1].CogY - points[counter].CogY, 2);
                results.Add(Math.Sqrt(dataX[counter] + dataY[counter]));
            }
            return results.Sum()/time;
        }


        public double ApEn(double[] data, int m, double r, int N, double std)
        {
            int Cm = 0, Cm1 = 0;
            double err = 0.0, sum = 0.0;

            err = std*r;

            for (int i = 0; i < N - (m + 1) + 1; i++)
            {
                Cm = Cm1 = 0;

                for (int j = 0; j < N - (m + 1) + 1; j++)
                {
                    bool eq = true;

                    for (int k = 0; k < m; k++)
                    {
                        if (Math.Abs(data[i + k] - data[j + k]) > err)
                        {
                            eq = false;
                            break;
                        }
                    }

                    if (eq) Cm++;

                    var k1 = m;
                    if (eq && Math.Abs(data[i + k1] - data[j + k1]) <= err)
```

```
                        Cm1++;
                }

                if (Cm > 0 && Cm1 > 0)
                    sum += Math.Log((double) Cm/(double) Cm1);
            }

            return sum/(double) (N - m);
        }

        public double SmEn(double[] data, int m, double r, int N, double std)
        {
            int Cm = 0, Cm1 = 0;
            double err = 0.0, sum = 0.0;

            err = std*r;

            for (int i = 0; i < N - (m + 1) + 1; i++)
            {
                for (int j = i + 1; j < N - (m + 1) + 1; j++)
                {
                    bool eq = true;
                    //m - length series
                    for (int k = 0; k < m; k++)
                    {
                        if (Math.Abs(data[i + k] - data[j + k]) > err)
                        {
                            eq = false;
                            break;
                        }
                    }
                    if (eq) Cm++;

                    //m+1 - length series
                    int k1 = m;
                    if (eq && Math.Abs(data[i + k1] - data[j + k1]) <= err)
                        Cm1++;
                }
            }

            if (Cm > 0 && Cm1 > 0)
                return Math.Log((double) Cm/(double) Cm1);
            else
                return 0.0;
        }
    }
}
```

BALANCE BOARD

Used to provide interaction with balance board

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
// file:models\balanceboard.cs
//
// summary:     Implements the balanceboard class
////////////////////////////////////////////////////////////////////////////////////////////////////
//

using System;
using WiimoteLib;

namespace VetsLegacy.Models.ExternalWrappers.BalanceBoard
{

////////////////////////////////////////////////////////////////////////////////////////////////////
//
    /// <summary>   A balance board. </summary>
    /// <remarks>   Maxim, 4/18/2014. </remarks>
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
    public class BalanceBoard
    {
        /// <summary>   The current balance board. </summary>
        private static Wiimote _balanceBoard;

        public BalanceBoard()
        {
            _balanceBoard = new Wiimote();
            _balanceBoard.WiimoteExtensionChanged += _balanceBoard_WiimoteExtensionChanged;
            Connect();
        }

        private void _balanceBoard_WiimoteExtensionChanged(object sender,
WiimoteExtensionChangedEventArgs e)
        {
            IsConnected = false;
        }


        public bool IsConnected { get; set; }

        public static BalanceBoard GetBoard()
        {
            return new BalanceBoard();
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Dispose current active balance board. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static void DisposeCurrent()
        {
            _balanceBoard.Disconnect();
            _balanceBoard.Dispose();
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets current point from balance board. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <returns>   The point. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public BalanceBoardPoint GetPoint()
        {
            try
            {
                return new BalanceBoardPoint
                {
                    CogX = (_balanceBoard.WiimoteState.BalanceBoardState.CenterOfGravity.X),
                    CogY = (_balanceBoard.WiimoteState.BalanceBoardState.CenterOfGravity.Y*-1),
                    TopLeft =
Round(_balanceBoard.WiimoteState.BalanceBoardState.SensorValuesLb.TopLeft),
                    TopRight =
Round(_balanceBoard.WiimoteState.BalanceBoardState.SensorValuesLb.TopRight),
                    BottomLeft =
Round(_balanceBoard.WiimoteState.BalanceBoardState.SensorValuesLb.BottomLeft),
```

```
                    BottomRight =
Round(_balanceBoard.WiimoteState.BalanceBoardState.SensorValuesLb.BottomRight)
                };
            }
            catch (Exception)
            {
                return new BalanceBoardPoint();
            }
        }

//////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Rounds number to 2 decimal places. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="number">   Number of. </param>
        ///
        /// <returns>   A double. </returns>

//////////////////////////////////////////////////////////////////////////////////////////////
//
        private double Round(double number)
        {
            return Math.Round(number, 2);
        }

//////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Connects balance board to program. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>

//////////////////////////////////////////////////////////////////////////////////////////////
//
        public void Connect()
        {
            try
            {
                _balanceBoard.Connect();
                _balanceBoard.SetLEDs(true, true, true, true);
                IsConnected = true;
            }
            catch
            {
                IsConnected = false;
            }
        }
    }
}
```

EXCEL READER

Used to provide access to excel sheets for reading from VETS

```
//////////////////////////////////////////////////////////////////////////////////////////////
//
// file:Models\Excel\PassByReader.cs
//
// summary:     Implements the pass by reader class
//////////////////////////////////////////////////////////////////////////////////////////////
//

using System;
using Microsoft.Office.Interop.Excel;

namespace VetsLegacy.Models.Readers
{
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
    /// <summary>   A pass by reader. </summary>
    /// <remarks>   Maxim, 5/15/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
    public class ExcelReader : IDisposable
    {
        /// <summary>   The column. </summary>
        private const int Column = 6;

        /// <summary>   Full pathname of the file. </summary>
        private readonly string _pathName;

        private Application _application;

        private Workbook _workbook;
        private Worksheet _worksheet;


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Constructor. </summary>
        /// <remarks>   Maxim, 5/15/2014. </remarks>
        /// <param name="fileName"> Filename of the file. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public ExcelReader(string fileName)
        {
            _pathName = fileName;
            SetWorksheet();
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>
        ///     Performs application-defined tasks associated with freeing, releasing, or
resetting unmanaged
        ///     resources.
        /// </summary>
        /// <remarks>   Maxim, 5/20/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public void Dispose()
        {
            _workbook.Close(null, null, null);
        }

        private void SetWorksheet()
        {
            _application = new Application();
            _workbook = _application.Workbooks.Open(_pathName);
            _worksheet = _workbook.Sheets[1];
        }

        public double GetValueForAnalysisValue(string analysis)
        {
            switch (analysis)
            {
                case "copSwayPCA":
                    return GetValue(2);
                case "copVelocity":
                    return GetValue(3);
                case "rmsX":
```

```
                    return GetValue(4);
                case "rmsY":
                    return GetValue(5);
                default:
                    return 0;
            }
        }

        private double GetValue(int row)
        {
            var range = (Range) _worksheet.Cells[row, Column];
            if (range != null && range.Value != null)
            {
                return range.Value;
            }
            return 0;
        }
    }
}
```

MICROTIMER

Used to provide timing system for trials

```
using System;

namespace VetsLegacy.Models.Timer
{
    /// <summary>
    /// MicroStopwatch class
    /// </summary>
    public class MicroStopwatch : System.Diagnostics.Stopwatch
    {
        readonly double _microSecPerTick =
            1000000D / System.Diagnostics.Stopwatch.Frequency;

        public MicroStopwatch()
        {
            if (!System.Diagnostics.Stopwatch.IsHighResolution)
            {
                throw new Exception("On this system the high-resolution " +
                                    "performance counter is not available");
            }
        }

        public long ElapsedMicroseconds
        {
            get
            {
                return (long)(ElapsedTicks * _microSecPerTick);
            }
        }
    }

    /// <summary>
    /// MicroTimer class
    /// </summary>
    public class MicroTimer
    {
        public delegate void MicroTimerElapsedEventHandler(
                            object sender,
                            MicroTimerEventArgs timerEventArgs);
        public event MicroTimerElapsedEventHandler MicroTimerElapsed;

        System.Threading.Thread _threadTimer;
        long _ignoreEventIfLateBy = long.MaxValue;
        long _timerIntervalInMicroSec = 0;
        bool _stopTimer = true;

        public MicroTimer()
        {
```

```
        }

        public MicroTimer(long timerIntervalInMicroseconds)
        {
            Interval = timerIntervalInMicroseconds;
        }

        public long Interval
        {
            get
            {
                return System.Threading.Interlocked.Read(
                    ref _timerIntervalInMicroSec);
            }
            set
            {
                System.Threading.Interlocked.Exchange(
                    ref _timerIntervalInMicroSec, value);
            }
        }

        public long IgnoreEventIfLateBy
        {
            get
            {
                return System.Threading.Interlocked.Read(
                    ref _ignoreEventIfLateBy);
            }
            set
            {
                System.Threading.Interlocked.Exchange(
                    ref _ignoreEventIfLateBy, value <= 0 ? long.MaxValue : value);
            }
        }

        public bool Enabled
        {
            set
            {
                if (value)
                {
                    Start();
                }
                else
                {
                    Stop();
                }
            }
            get
            {
                return (_threadTimer != null && _threadTimer.IsAlive);
            }
        }

        public void Start()
        {
            if (Enabled || Interval <= 0)
            {
                return;
            }

            _stopTimer = false;

            System.Threading.ThreadStart threadStart = () => NotificationTimer(ref
_timerIntervalInMicroSec,
                ref _ignoreEventIfLateBy,
                ref _stopTimer);
```

```
            _threadTimer = new System.Threading.Thread(threadStart) {Priority =
System.Threading.ThreadPriority.Highest};
            _threadTimer.Start();
        }

        public void Stop()
        {
            _stopTimer = true;
        }

        public void StopAndWait()
        {
            StopAndWait(System.Threading.Timeout.Infinite);
        }

        public bool StopAndWait(int timeoutInMilliSec)
        {
            _stopTimer = true;

            if (!Enabled || _threadTimer.ManagedThreadId ==
                System.Threading.Thread.CurrentThread.ManagedThreadId)
            {
                return true;
            }

            return _threadTimer.Join(timeoutInMilliSec);
        }

        public void Abort()
        {
            _stopTimer = true;

            if (Enabled)
            {
                _threadTimer.Abort();
            }
        }

        void NotificationTimer(ref long timerIntervalInMicroSec,
                               ref long ignoreEventIfLateBy,
                               ref bool stopTimer)
        {
            int  timerCount = 0;
            long nextNotification = 0;

            MicroStopwatch microStopwatch = new MicroStopwatch();
            microStopwatch.Start();

            while (!stopTimer)
            {
                long callbackFunctionExecutionTime =
                    microStopwatch.ElapsedMicroseconds - nextNotification;

                long timerIntervalInMicroSecCurrent =
                    System.Threading.Interlocked.Read(ref timerIntervalInMicroSec);
                long ignoreEventIfLateByCurrent =
                    System.Threading.Interlocked.Read(ref ignoreEventIfLateBy);

                nextNotification += timerIntervalInMicroSecCurrent;
                timerCount++;
                long elapsedMicroseconds = 0;

                while ((elapsedMicroseconds = microStopwatch.ElapsedMicroseconds)
                        < nextNotification)
                {
                    System.Threading.Thread.SpinWait(1);
                }

                long timerLateBy = elapsedMicroseconds - nextNotification;
```

```
                if (timerLateBy >= ignoreEventIfLateByCurrent)
                {
                    continue;
                }

                MicroTimerEventArgs microTimerEventArgs =
                     new MicroTimerEventArgs(timerCount,
                                             elapsedMicroseconds,
                                             timerLateBy,
                                             callbackFunctionExecutionTime);
                MicroTimerElapsed(this, microTimerEventArgs);
            }

            microStopwatch.Stop();
        }
    }

    /// <summary>
    /// MicroTimer Event Argument class
    /// </summary>
    public class MicroTimerEventArgs : EventArgs
    {
        // Simple counter, number times timed event (callback function) executed
        public int  TimerCount { get; private set; }

        // Time when timed event was called since timer started
        public long ElapsedMicroseconds { get; private set; }

        // How late the timer was compared to when it should have been called
        public long TimerLateBy { get; private set; }

        // Time it took to execute previous call to callback function (OnTimedEvent)
        public long CallbackFunctionExecutionTime { get; private set; }

        public MicroTimerEventArgs(int  timerCount,
                                   long elapsedMicroseconds,
                                   long timerLateBy,
                                   long callbackFunctionExecutionTime)
        {
            TimerCount = timerCount;
            ElapsedMicroseconds = elapsedMicroseconds;
            TimerLateBy = timerLateBy;
            CallbackFunctionExecutionTime = callbackFunctionExecutionTime;
        }
    }
}
```

RESULT

Used to provide result container object for results system

```
//////////////////////////////////////////////////////////////////////////////////////////////////
//
// file:models\result.cs
//
// summary:     Implements the result class
//////////////////////////////////////////////////////////////////////////////////////////////////
//

namespace VetsLegacy.Models.User.Results
{

//////////////////////////////////////////////////////////////////////////////////////////////////
//
    /// <summary>   A result. </summary>
    /// <remarks>   Maxim, 4/25/2014. </remarks>

//////////////////////////////////////////////////////////////////////////////////////////////////
//
```

```
    public class Result
    {

//////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the type of the category. </summary>
        ///
        /// <value> The type of the category. </value>

//////////////////////////////////////////////////////////////////////////////////////////////////
//

        public string CategoryType { get; set; }


//////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the point x coordinate. </summary>
        ///
        /// <value> The point x coordinate. </value>

//////////////////////////////////////////////////////////////////////////////////////////////////
//

        public double PointX { get; set; }

    }
}
```

SETTINGS

Used to provide settings model for VETS

```
//////////////////////////////////////////////////////////////////////////////////////////////////
//
// file:models\settings.cs
//
// summary:     Implements the settings class
//////////////////////////////////////////////////////////////////////////////////////////////////
//

using System;

namespace VetsLegacy.Models.User.Settings
{

//////////////////////////////////////////////////////////////////////////////////////////////////
//
    /// <summary>   A settings. </summary>
    /// <remarks>   Maxim, 4/25/2014. </remarks>

//////////////////////////////////////////////////////////////////////////////////////////////////
//

    public class Settings
    {
        /// <summary>   The current settings. </summary>
        private static Settings _currentSettings;


//////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the sampling rate. </summary>
        ///
        /// <value> The sampling rate. </value>

//////////////////////////////////////////////////////////////////////////////////////////////////
//
```

```
        public double SamplingRate { get; set; }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the wait time between sessions. </summary>
        ///
        /// <value> The wait time between sessions. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//

        public double WaitTimeBetweenSessions { get; set; }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the time per trial. </summary>
        ///
        /// <value> The time per trial. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//

        public double TimePerTrial { get; set; }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Sets current setting as default. </summary>
        ///
        /// <remarks>   Maxim, 4/21/2014. </remarks>
        ///
        /// <param name="settings"> Options for controlling the operation. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//

        public static void SetCurrentSettingAsDefault(Settings settings)
        {
            _currentSettings = settings;
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets current settings. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <returns>   The current settings. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//

        public static Settings GetCurrentSettings()
        {
           // var filePathToSettingsXmlFile =
Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)+"/Vets/Config/settings.xml";
            if (_currentSettings != null)
            {
                return _currentSettings;
            }
            return new Settings
            {
                SamplingRate = 100,
                TimePerTrial = 60,
                WaitTimeBetweenSessions = 30
            };
```

```
        }
    }
}
```

TRIAL

Used to provide container object for a Trial

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
// file:models\trial.cs
//
// summary:     Implements the trial class
////////////////////////////////////////////////////////////////////////////////////////////////////
//

using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Windows;
using System.Windows.Input;
using Syncfusion.Windows.Shared;
using Syncfusion.XlsIO.Implementation.Collections;
using VetsLegacy.Models.ExternalWrappers.BalanceBoard;

namespace VetsLegacy.Models.User.Trial
{

////////////////////////////////////////////////////////////////////////////////////////////////////
//
    /// <summary>   A trial. </summary>
    /// <remarks>   Maxim, 4/25/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
    public class Trial
    {
        /// <summary>   The trials. </summary>
        public static List<Trial> Trials = new List<Trial>();

        /// <summary>   The counter. </summary>
        public static int Counter
        {
            get
            {
                if (Trials.Count == 0)
                {
                    return 1;
                }
                return Trials.Count + 1;
            }
            set { _counter = value; }
        }

        /// <summary>   The points. </summary>
        public List<BalanceBoardPoint> Points = new List<BalanceBoardPoint>();

        private static int _counter;


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Constructor. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="type">     The type. </param>
        /// <param name="option">   . </param>
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public Trial(TrialType type, TrialOption option)
        {
            SessionType = type;
            TrialOption = option;
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Constructor. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="name">     The name. </param>
        /// <param name="command">  The command. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public Trial(string name, ICommand command)
        {
            switch (name)
            {
                case "bFirm":
                    SessionType = TrialType.Black;
                    TrialOption = TrialOption.Firm;
                    Delete = command;
                    SessionName = "Eyes Closed";

                    break;
                case "bFoam":
                    SessionType = TrialType.Black;
                    TrialOption = TrialOption.Foam;
                    Delete = command;
                    SessionName = "Eyes Closed";
                    break;
                case "dFirm":
                    SessionType = TrialType.Dynamic;
                    TrialOption = TrialOption.Firm;
                    Delete = command;
                    SessionName = "Dynamic";
                    break;
                case "dFoam":
                    SessionType = TrialType.Dynamic;
                    TrialOption = TrialOption.Foam;
                    Delete = command;
                    SessionName = "Dynamic";
                    break;
                case "sFirm":
                    SessionType = TrialType.Static;
                    TrialOption = TrialOption.Firm;
                    Delete = command;
                    SessionName = "Eyes Open";
                    break;
                case "sFoam":
                    SessionType = TrialType.Static;
                    TrialOption = TrialOption.Foam;
                    Delete = command;
                    SessionName = "Eyes Open";
                    break;
            }
            Priority = Counter;
        }
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the trial option. </summary>
        ///
        /// <value> The trial option. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public TrialOption TrialOption { get; set; }


////////////////////////////////////////////////////////////////////////////////////////////////////
//

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the type of the session. </summary>
        ///
        /// <value> The type of the session. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public TrialType SessionType { get; set; }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the delete commad for trial. </summary>
        ///
        /// <value> The delete. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public ICommand Delete { get; set; }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the priority of the trial. </summary>
        ///
        /// <value> The priority. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public int Priority { get; set; }

        public int Interval { get; set; }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the name of the session. </summary>
        ///
        /// <value> The name of the session. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public string SessionName { get; set; }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Clears this object to its blank/initial state. </summary>
        ///
```

```
        /// <remarks>   Maxim, 5/15/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static void Clear()
        {
            Trials.Clear();
        }

        public static void AddTrialToList(Trial trial)
        {
            Trials.Add(trial);
            ResetNumbers();
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Resets the trial with described by trial. </summary>
        ///
        /// <remarks>   Maxim, 5/7/2014. </remarks>
        ///
        /// <param name="trial">    The trial. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static void ResetTrialWith(List<Trial> trial)
        {
            Trials.Clear();
            Trials.AddRange(trial);
            ResetNumbers();
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Adds the trials to list. </summary>
        ///
        /// <remarks>   Maxim, 4/28/2014. </remarks>
        ///
        /// <param name="trial">    The trial. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static void AddTrialsToList(ConcurrentQueue<Trial> trial)
        {
            Trials.AddRange(trial.ToList());
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Removes the trial at described by number. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="number">   Number of. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static void RemoveTrialAt(int number)
        {
            if (number < Trials.Count)
            {
                Trials.RemoveAt(number);
            }
            ResetNumbers();
        }
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Returns trial as queue. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <returns>   The trial as queue. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static ConcurrentQueue<Trial> ReturnTrialAsQueue()
        {
            return new ConcurrentQueue<Trial>(Trials);
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Returns trials as list. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <returns>   The trials as list. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static List<Trial> ReturnTrialsAsList()
        {
            return Trials.ToList();
        }

        public static List<Trial> ReturnStandard()
        {
            //3 eyes-open firm, 3 eyes-closed firm, 3 dynamic firm, 3 eyes-open foam, 3 eyes-
closed foam, 3 dynamic foam. This button should be called "Standard Order"
            var returnList = new SFArrayList<Trial>();
            var eoFirm1 = new Trial(TrialType.Static, TrialOption.Firm)
            {
                SessionName = "Eyes Open"
            };
            var eoFirm2 = new Trial(TrialType.Static, TrialOption.Firm)
            {
                SessionName = "Eyes Open"
            };
            var eoFirm3 = new Trial(TrialType.Static, TrialOption.Firm)
            {
                SessionName = "Eyes Open"
            };
            var ecFirm1 = new Trial(TrialType.Black, TrialOption.Firm)
            {
                SessionName = "Eyes Closed"
            };
            var ecFirm2 = new Trial(TrialType.Black, TrialOption.Firm)
            {
                SessionName = "Eyes Closed"
            };
            var ecFirm3 = new Trial(TrialType.Black, TrialOption.Firm)
            {
                SessionName = "Eyes Closed"
            };
            var dynFirm1 = new Trial(TrialType.Dynamic, TrialOption.Firm)
            {
                SessionName = "Dynamic"
            };
            var dynFirm2 = new Trial(TrialType.Dynamic, TrialOption.Firm)
            {
                SessionName = "Dynamic"
            };
```

```
            var dynFirm3 = new Trial(TrialType.Dynamic, TrialOption.Firm)
            {
                SessionName = "Dynamic"
            };
            var eoFoam1 = new Trial(TrialType.Static, TrialOption.Foam)
            {
                SessionName = "Eyes Open"
            };
            var eoFoam2 = new Trial(TrialType.Static, TrialOption.Foam)
            {
                SessionName = "Eyes Open"
            };
            var eoFoam3 = new Trial(TrialType.Static, TrialOption.Foam)
            {
                SessionName = "Eyes Open"
            };
            var ecFoam1 = new Trial(TrialType.Black, TrialOption.Foam)
            {
                SessionName = "Eyes Closed"
            };
            var ecFoam2 = new Trial(TrialType.Black, TrialOption.Foam)
            {
                SessionName = "Eyes Closed"
            };
            var ecFoam3 = new Trial(TrialType.Black, TrialOption.Foam)
            {
                SessionName = "Eyes Closed"
            };
            var dynFoam1 = new Trial(TrialType.Dynamic, TrialOption.Foam)
            {
                SessionName = "Dynamic"
            };
            var dynFoam2 = new Trial(TrialType.Dynamic, TrialOption.Foam)
            {
                SessionName = "Dynamic"
            };
            var dynFoam3 = new Trial(TrialType.Dynamic, TrialOption.Foam)
            {
                SessionName = "Dynamic"
            };
```

```
            eoFirm1.Priority = 1;
            eoFirm2.Priority = 2;
            eoFirm3.Priority = 3;
            ecFirm1.Priority = 4;
            ecFirm2.Priority = 5;
            ecFirm3.Priority = 6;
            dynFirm1.Priority = 7;
            dynFirm2.Priority = 8;
            dynFirm3.Priority = 9;
            eoFoam1.Priority = 10;
            eoFoam2.Priority = 11;
            eoFoam3.Priority = 12;
            ecFoam1.Priority = 13;
            ecFoam2.Priority = 14;
            ecFoam3.Priority = 15;
            dynFoam1.Priority = 16;
            dynFoam2.Priority = 17;
            dynFoam3.Priority = 18;
            returnList.Add(eoFirm1);
            returnList.Add(eoFirm2);
            returnList.Add(eoFirm3);
            returnList.Add(ecFirm1);
            returnList.Add(ecFirm2);
            returnList.Add(ecFirm3);
            returnList.Add(dynFirm1);
            returnList.Add(dynFirm2);
            returnList.Add(dynFirm3);
            returnList.Add(eoFoam1);
```

```
            returnList.Add(eoFoam2);
            returnList.Add(eoFoam3);
            returnList.Add(ecFoam1);
            returnList.Add(ecFoam2);
            returnList.Add(ecFoam3);
            returnList.Add(dynFoam1);
            returnList.Add(dynFoam2);
            returnList.Add(dynFoam3);
            return returnList;
        }

        public static List<Trial> ReturnRandom()
        {
            var returnList = new List<Trial>();
            var eoFirm1 = new Trial(TrialType.Static, TrialOption.Firm)
            {
                SessionName = "Eyes Open",
                Priority = 1
            };
            var ecFirm2 = new Trial(TrialType.Black, TrialOption.Firm)
            {
                SessionName = "Eyes Closed",
                Priority = 2
            };
            returnList.Add(eoFirm1);
            returnList.Add(ecFirm2);

            var fullTrialList = new List<Trial>
            {
                new Trial(TrialType.Black, TrialOption.Firm){ SessionName = "Eyes Closed"},
                new Trial(TrialType.Black, TrialOption.Firm){ SessionName = "Eyes Closed"},
                new Trial(TrialType.Black, TrialOption.Foam)  { SessionName = "Eyes Closed"},
                new Trial(TrialType.Black, TrialOption.Foam)  { SessionName = "Eyes Closed"},
                new Trial(TrialType.Black, TrialOption.Foam)  { SessionName = "Eyes Closed"},
                new Trial(TrialType.Static, TrialOption.Firm){ SessionName = "Eyes Open" },
                new Trial(TrialType.Static, TrialOption.Firm){ SessionName = "Eyes Open" },
                new Trial(TrialType.Static, TrialOption.Foam){SessionName = "Eyes Open"},
                new Trial(TrialType.Static, TrialOption.Foam){SessionName = "Eyes Open"},
                new Trial(TrialType.Static, TrialOption.Foam){SessionName = "Eyes Open"},
                new Trial(TrialType.Dynamic, TrialOption.Firm){   SessionName = "Dynamic"},
                new Trial(TrialType.Dynamic, TrialOption.Firm){   SessionName = "Dynamic"},
                new Trial(TrialType.Dynamic, TrialOption.Firm){   SessionName = "Dynamic"},
                new Trial(TrialType.Dynamic, TrialOption.Foam)  {  SessionName = "Dynamic"},
                new Trial(TrialType.Dynamic, TrialOption.Foam)  {  SessionName = "Dynamic"},
                new Trial(TrialType.Dynamic, TrialOption.Foam)  {  SessionName = "Dynamic"}
            };
            List<Trial> randomList = fullTrialList.OrderBy(m => Guid.NewGuid()).ToList();
            var counter = 3;
            foreach (var trial in randomList)
            {
                trial.Priority = counter;
                counter++;
            }
            returnList.AddRange(randomList);
            return returnList;
        }

////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Resets the numbers(used in trial list in settings) </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static void ResetNumbers()
        {
            for (int counter = 0; counter < Trials.Count; counter++)
```

```
            {
                Trials[counter].Priority = counter + 1;
            }
        }

        public static List<Trial> ResetNumbers(List<Trial> trials)
        {
            var counter = 1;
            var temp = trials;
            foreach (var trial in temp)
            {
                trial.Priority = counter;
                Console.WriteLine(trial.Priority);
                counter++;
            }
            return temp;
        }

////////////////////////////////////////////////////////////////////////////////////////////////
//

////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Adds the points to list. </summary>
        /// <summary>   Adds the points to list. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        /// <remarks>   Maxim, 4/28/2014. </remarks>
        ///
        /// <param name="points">   The points. </param>

////////////////////////////////////////////////////////////////////////////////////////////////
//
        public void AddPointsToList(List<BalanceBoardPoint> points)
        {
            foreach (BalanceBoardPoint point in points)
            {
                Points.Add(point);
            }
        }
    }
}
```

TRIAL OPTIONS

Used to provide trial options for VETS

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// file:Models\Trial\Options.cs
//
// summary:     Implements the options class
////////////////////////////////////////////////////////////////////////////////////////////////
//

namespace VetsLegacy.Models.User.Trial
{

////////////////////////////////////////////////////////////////////////////////////////////////
//
    /// <summary>   Values that represent TrialType. </summary>
    ///
    /// <remarks>   Maxim, 4/25/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////
//

    public enum TrialOption
```

```
    {
        /// <summary>   An enum constant representing the foam option. </summary>
        Foam,

        /// <summary>   An enum constant representing the firm option. </summary>
        Firm
    }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
    /// <summary>   Values that represent TrialType. </summary>
    ///
    /// <remarks>   Maxim, 5/7/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//

    public enum TrialType
    {
        /// <summary>   An enum constant representing the black option. </summary>
        Black,

        /// <summary>   An enum constant representing the static option. </summary>
        Static,

        /// <summary>   An enum constant representing the dynamic option. </summary>
        Dynamic
    }
}
```

PATIENT

Used to provide patient object for storing patients

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
// file:Models\Patient.cs
//
// summary:     Implements the patient class
////////////////////////////////////////////////////////////////////////////////////////////////////
//

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Windows.Input;

namespace VetsLegacy.Models.User
{

////////////////////////////////////////////////////////////////////////////////////////////////////
//
    /// <summary>   Patient Class. </summary>
    /// <remarks>   Maxim, 4/18/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
    public class Patient
    {
        private static string _pathToMostRecentItem;

        /// <summary>   The patients. </summary>
        public static List<Patient> Patients = new List<Patient>();

        public static string GetDirectoryPathWithPatientName(string name)
        {
            return Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) +
@"/Vets/Results/" + name;
        }
```

```
        public static void SetDirectoryNameOfMostRecent(string name)
        {
            var parentPath = String.Empty;
            var directories = name.Split('\\');
            var directoryLength = directories.Length;
            for (int counter = 0; counter < directoryLength - 2; counter++)
            {
                parentPath += directories[counter]+@"\";
            }
            _pathToMostRecentItem = parentPath;
        }

        public static string GetDirectoryNameOfMostRececnt()
        {
            if (_pathToMostRecentItem != null)
            {
                return _pathToMostRecentItem;
            }
            return String.Empty;
        }

        /// <summary>   Full pathname of the file. </summary>
        private static string _filePath;


////////////////////////////////////////////////////////////////////////////////////////////////////
//

        /// <summary>   Gets or sets the trials. </summary>
        public List<Trial.Trial> Trials = new List<Trial.Trial>();

        /// <summary>   Gets or sets the name of the patient. </summary>
        /// <value> The name of the patient. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets the name of the patient. </summary>
        ///
        /// <value> The name of the patient. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public string PatientName { get; set; }

        /// <summary>   Gets or sets information describing the render new. </summary>
        /// <value> Information describing the render new. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets or sets information describing the render new. </summary>
        ///
        /// <value> Information describing the render new. </value>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public ICommand RenderNewData { get; set; }

        /// <summary>   Event queue for all listeners interested in PropertyChanged events.
</summary>
        public event PropertyChangedEventHandler PropertyChanged;
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Adds the trials. </summary>
        ///
        /// <remarks>   Maxim, 5/7/2014. </remarks>
        ///
        /// <param name="trials">   The trials. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public void AddTrials(List<Trial.Trial> trials)
        {
            Trials.AddRange(trials);
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Adds a trial. </summary>
        ///
        /// <remarks>   Maxim, 5/7/2014. </remarks>
        ///
        /// <param name="trial">    The trial. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public void AddTrial(Trial.Trial trial)
        {
            Trials.Add(trial);
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets the trials. </summary>
        ///
        /// <remarks>   Maxim, 5/7/2014. </remarks>
        ///
        /// <returns>   The trials. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public List<Trial.Trial> GetTrials()
        {
            return Trials;
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Adds a patient. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="patient">  The patient. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static void AddPatient(Patient patient)
        {
            _filePath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments) +
@"/Vets/Results/";
            if (Patients.Exists(m => m.PatientName == patient.PatientName))
```

```
            {
                Patient p = Patients.FirstOrDefault(m => m.PatientName == patient.PatientName);
                Patients[Patients.IndexOf(p)] = patient;
            }
            else
            {
                Patients.Add(patient);
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Adds a patient. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="name"> The name. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static void AddPatient(string name)
        {
            var patient = new Patient
            {
                PatientName = name
            };
            AddPatient(patient);
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets the last added patient. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <returns>   The last added patient. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static Patient GetLastAddedPatient()
        {
            return Patients[Patients.Count - 1];
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Query if 'name' is valid patient. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="name"> The name. </param>
        ///
        /// <returns>   true if valid patient, false if not. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static bool IsValidPatient(string name)
        {
            return Patients.Exists(m => m.PatientName == name);
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Adds a trial to known patient. </summary>
        ///
```

```
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="patient">  The patient. </param>
        /// <param name="trial">    The trial. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static void AddTrialToPatient(Patient patient, Trial.Trial trial)
        {
            if (Patients.Contains(patient))
            {
                Patients[Patients.IndexOf(patient)].Trials.Add(trial);
            }
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Adds a multiple trials to patient to 'trials'. </summary>
        ///
        /// <remarks>   Maxim, 4/27/2014. </remarks>
        ///
        /// <param name="patient">  The patient. </param>
        /// <param name="trials">   The trials. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static void AddMultipleTrialsToPatient(Patient patient, List<Trial.Trial> trials)
        {
            if (Patients.Contains(patient))
            {
                Patients[Patients.IndexOf(patient)].Trials.AddRange(trials);
            }
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Adds a trial to patient from name of patient. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="name">     The name. </param>
        /// <param name="trial">    The trial. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static void AddTrialToPatient(string name, Trial.Trial trial)
        {
            if (!Patients.Exists(p => p.PatientName == name)) return;
            foreach (Patient patient in Patients)
            {
                if (patient.PatientName == name)
                {
                    patient.Trials.Add(trial);
                }
            }
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets patient from string. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="name"> The name. </param>
        ///
        /// <returns>   The patient from string. </returns>
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static Patient GetPatientFromString(string name)
        {
            return Patients.FirstOrDefault(t => t.PatientName == name);
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets patients as list. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <returns>   The patients as list. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static List<Patient> GetPatientsAsList()
        {
            return Patients.ToList();
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets trials for patient. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="patient">  The patient. </param>
        ///
        /// <returns>   The trials for patient. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static List<Trial.Trial> GetTrialsForPatient(Patient patient)
        {
            if (Patients.Contains(patient))
            {
                return patient.Trials;
            }
            return null;
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets trials for patient. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="patient">  The patient. </param>
        ///
        /// <returns>   The trials for patient. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static List<Trial.Trial> GetTrialsForPatient(string patient)
        {
            return GetTrialsForPatient(Patients.FirstOrDefault(t => t.PatientName == patient));
        }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Deletes the patient described by patient. </summary>
        ///
```

```
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <param name="patient">  The patient. </param>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static void DeletePatient(Patient patient)
        {
            if (Patients.Contains(patient))
            {
                Patients.Remove(patient);
            }
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Clears this object to its blank/initial state. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public static void Clear()
        {
            Patients.Clear();
        }


////////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Gets trial count. </summary>
        ///
        /// <remarks>   Maxim, 4/18/2014. </remarks>
        ///
        /// <returns>   The trial count. </returns>

////////////////////////////////////////////////////////////////////////////////////////////////////
//
        public int GetTrialCount()
        {
            return Trials.Count;
        }
    }
}
```

EXCEL WRITER

Used to provide functionality to write to excel

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
// file:Models\Excel\PassByWriter.cs
//
// summary:     Implements the pass by writer class
////////////////////////////////////////////////////////////////////////////////////////////////////
//

using System;
using System.Collections.Generic;
using System.Linq;
using AForge;
using Syncfusion.XlsIO;
using Syncfusion.XPS;
using VetsLegacy.Models.Analysis;
using VetsLegacy.Models.ExternalWrappers;
using VetsLegacy.Models.ExternalWrappers.BalanceBoard;

namespace VetsLegacy.Models.Writers
{
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
    /// <summary>   A pass by writer. </summary>
    /// <remarks>   Maxim, 5/14/2014. </remarks>

////////////////////////////////////////////////////////////////////////////////////////////////
//
    public class ExcelWriter : IDisposable
    {
        public void Dispose()
        {
        }


////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Writes a cop sway to file. </summary>
        ///
        /// <remarks>   Maxim, 5/14/2014. </remarks>
        ///
        /// <param name="worksheet">    The worksheet. </param>
        /// <param name="points">       The points. </param>

////////////////////////////////////////////////////////////////////////////////////////////////
//
        public void WriteCopSwayToFile(IWorksheet worksheet, List<BalanceBoardPoint> points)
        {
            var pts = new double[points.Count, 2];
            for (int counter = 0; counter < points.Count; counter++)
            {
                pts[counter, 0] = points[counter].CogX;
                pts[counter, 1] = points[counter].CogY;
            }
            var copSway = new ClassCalculations().GetCopSwayAreaPca(pts);
            Write(worksheet, "CopSwayAreaPCA", copSway, 5, 6, 2);
        }


////////////////////////////////////////////////////////////////////////////////////////////////
//
        /// <summary>   Writes a cop velocity to file. </summary>
        ///
        /// <remarks>   Maxim, 5/14/2014. </remarks>
        ///
        /// <param name="worksheet">    The worksheet. </param>
        /// <param name="points">       The points. </param>

////////////////////////////////////////////////////////////////////////////////////////////////
//
        public void WriteCopVelocityToFile(IWorksheet worksheet, List<BalanceBoardPoint>
points,double sampleRate,double time)
        {
            var copVelocity = new
ClassCalculations().GetCopVelocitySampleRate(points,sampleRate,time);
            Write(worksheet, "CopVelocity", copVelocity, 5, 6, 3);
        }

        private void Write(IWorksheet worksheet, string name, double value, int nameColumn, int
valueColumn, int row)
        {
            worksheet.SetText(row, nameColumn, name);
            worksheet.SetNumber(row, valueColumn, Double.IsNaN(value) ? 0 : value);
        }

        public void WriteSampenToFile(IWorksheet worksheet, List<BalanceBoardPoint> points)
        {
        }
```

```
        public void WriteRmsToFile(IWorksheet worksheet, List<BalanceBoardPoint> points)
        {
            var point = GetRms(points);
            var rmsX = point.CogX;
            var rmsY = point.CogY;

            Write(worksheet, "Rms X", rmsX, 5, 6, 4);
            Write(worksheet, "Rms Y", rmsY, 5, 6, 5);
        }

        private BalanceBoardPoint GetRms(List<BalanceBoardPoint> listDataPoint)
        {
            List<double> listX = listDataPoint.Select(m => m.CogX).ToList();
            List<double> listY = listDataPoint.Select(m => m.CogY).ToList();

            double listMeansXAverage = listX.Average();
            double listMeansYAverage = listY.Average();

            var xValues = listX.Select(x => Math.Pow(listMeansXAverage - x, 2)).ToList();
            var yValues = listY.Select(y => Math.Pow(listMeansYAverage - y, 2)).ToList();
            var rmsX = Math.Sqrt(xValues.Average());
            var rmsY = Math.Sqrt(yValues.Average());
            return new BalanceBoardPoint
            {
                CogX = rmsX,
                CogY = rmsY,
            };
        }
    }
}
```

APPENDIX 11 - SAFETY PROGRAM PLAN

Table of Contents

Part **Page**

APPENDIX 11 - SAFETY PROGRAM PLAN

1. Introduction

This appendix contains a description of the requirements, forms, approvals and assurances relating to safety in the research environment. To ease the burden of submitting general institution safety program information with each proposal, the USAMRMC has developed a Facility Safety Plan program. If you have any questions concerning this appendix, please contact Ms. Cavelle Williams of the USAMRMC Safety Office at 301-619-6035 or email at Cavelle.Williams@det.amedd.army.mil.

A Facility Safety Plan is a 2-10 page document that summarizes the institution's safety program. **Approval of the Facility Safety Plan is granted on an institution basis rather than on a proposal basis.** The Facility Safety Plan shall be institution-based, consist of six parts as outlined the **Facility Safety Plan**, part 2 of this appendix, and be prepared by the Facility Safety Director/Manager of the institution. Each institution is required to submit only one Facility Safety Plan. An institution with multiple research sites, subcontractor or a consortium must submit a separate Facility Safety Plan for each research site. The Facility Safety Plan submission for each site will include signed assurances from both the Facility Safety Manager and Principal Investigator Assurance (part 2F of this appendix).

Facility Safety Plan approvals are granted for a 5-year period with annual updates required (part 4 of this appendix). To determine if your organization has an approved Facility Safety Plan, check our website listing at: https://mrmc.detrick.army.mil/crprcqsohdfsplan.asp

a. If your organization's name **appears** on this Institutional Facility Safety Plan listing **and** approval of the Facility Safety Plan has not expired, then your institution's Facility Safety Plan need not be sent with the proposal submittal.

b. If either your organization's name **does not appear** on this Institutional Facility Safety Plan listing **or** the approval of your institution's Facility Safety Plan has expired, your Facility Safety Manager/Director must provide the U.S. Army Medical Research and Materiel Command's (USAMRMC's) Safety Office with a Facility Safety Plan and a signed assurance, as outlined below (part 2 of this appendix).

2. Facility Safety Plan (Institution-Based)

The Facility Safety Director/Manager must provide information from the institutional perspective, as appropriate, for each of the six parts listed below. **This Facility Safety Plan should not reference the specific proposal**. A list of the first five components with a brief description of each is acceptable. **Do not send** institution safety manuals, although they may be referenced in your submission (a web site address is also acceptable). **Do not label** "Not Applicable" or "N/A". Each element shown below of the Facility Safety Plan should be addressed by providing a statement as it applies to your institution as a whole. **Example:** (see Radioactive Materials, part c) If your institution does not use Radioactive Materials and does not have a copy of the Nuclear Regulatory Commission (NRC), state-approved license, or the equivalent in cases of institutions outside the continental US then provide a statement to that effect.

a. Research Operations/Standard Operating Procedures (SOPs)
Provide a brief description of the safety procedures relating to the medical research operation of the facility. These should include (a) a description of any special skills, training and SOPs that assure safe research operations (Bio-Safety Committee, Radiation Committee, HAZCOM, Blood-borne Pathogens, Chemical Hygiene Plan, etc.) and (b) a description of medical surveillance and support.

APPENDIX 11 - SAFETY PROGRAM PLAN

b. Facility Equipment and Description (Related to the Research Environment)
Provide (a) a description of the facility; (b) a description of personal protective equipment used within the facility; and a list of specialized safety equipment such as bio-safety cabinets, hoods, exhausts, and ventilation systems.

c. Radioactive Materials
Provide a current copy of the Nuclear Regulatory Commission or state-approved license.

d. Hazard Analysis (Related to the Research Environment)
Provide a description of each hazard identified, the hazard analysis performed based on maximum credible event and the plan to minimize or eliminate each hazard and control risk to laboratory personnel.

e. Biological Defense Research Program Requirements
(Only applicable to the Biological Defense Research Program funded awards). For those institutions where Principal Investigators are supported by the USAMRMC and are conducting research with Bio-safety Levels 3 and 4 material, a Facility Safety Plan must be prepared in accordance with 32 Code of Federal Regulations (CFR) 626.18. See the following URL: www.access.gpo.gov/nara/cfr/waisidx_99/32cfr626_99.html for a copy of the 32 CFR 626.18, Biological Defense Safety Program.

f. Facility Safety Director/Manager Assurance
The Facility Safety Director/Manager must provide the following signed assurance:

APPENDIX 11 - SAFETY PROGRAM PLAN

Facility Safety Director/Manager Assurance

- I assure that this institution has an existing institutional safety and occupational health program that meets appropriate Federal, Sate and local regulations as required by law, as well as the National Institute of Health Guidelines for Research Involving DNA Molecules, dated Jan 2001.
- I assure that all hazards associated with the research laboratories have been identified, eliminated and/or controlled in such a manner as to provide for a safe research laboratory environment.
- I accept full responsibility for submitting the annual **Facility Safety Plan Status Report** including significant changes in facility, safety equipment, and safety procedures by fax to 301-619-6627, by e-mail to cavelle.williams@det.amedd.army.mil, by mail to

  Commanding General,
  U.S. Army Medical Research and Materiel Command,
  ATTN: MCMR-ZC-SSE
  504 Scott Street
  Fort Detrick, MD 21702-5012.

- I assure that I have consulted with all current PI's holding USAMRMC awards concerning this institution's safety policies and procedures and will consult with all future PI's holding USAMRMC awards concerning this institution' s safety policies and procedures.
- Use of etiologic agents as defined in 32 CFR 626? ☒ Yes ☐ No
  "Etiologic agent = a viable microorganism, or its toxin which causes or may cause human disease, and includes those agents listed in 42 CFR 72.3 of the Dept of HHS regulations. **AND** any material of biological origin that poses a degree of hazard similar to those organisms.

Dr. Lily Lodhi

Name of Institution's Safety Director/Manager (print)

Signature

10/17/2014
Date

MAILING ADDRESS: 3307 North Broad St. Room B-49
Street

Philadelphia PA 19460
City State Zip Code

PHONE NUMBER: 215-707-2520

FAX: 215-707-1600

E-MAIL ADDRESS: lily.lodhi@temple.edu

WEB SITE: www.temple.edu/ehrs

Principal Investigator Assurance

♦ I assure that I have involved the Facility Safety Director/Manager in the planning of this research proposal, discussed with him/her all aspects of the proposal that relate to occupational health and safety, and will help him/her prepare the annual Facility Safety Plan Status Report.

♦ I assure that I will comply with my institution's safety program and its requirements.

♦ I understand that I am directly responsible for all aspects of safety and occupational health specific to my research protocol.

♦ I assure that I will report to the Facility Safety Director/Manager any changes in the safety or occupational health practices due to changes in my originally planned research.

♦ I assure that hazards associated with my research have been identified eliminated and/or controlled.

♦ I assure that all Facility Safety Plan requirements are in compliance with Local, State and Federal general industry standards.

♦ If applicable, I assure Biological research programs will follow the recommended guidelines established in the latest editions of the CDC-NIH publication Biosafety in Microbiological and Biomedical Laboratories (BMBL); Army Regulation 385-10, Chapter 20 (Biological Safety); and DA Pamphlet 385-69 (Safety Standards for Microbiological and Biomedical Laboratories).

♦ Use of Infectious Agents and Toxins (IAT) as defined below: ☐ Yes ☒ No

"Infectious Agent or Toxin = a viable microorganism, or its toxin which causes or may cause human disease, and includes those agents and includes those agents classified as Risk Group 2 or higher as defined in the latest edition of the Biosafety in Microbiological and Biomedical Laboratories (BMBL)."

W. Geoffrey Wright
Name of Principal Investigator (print)

WGWright
Signature

15 Oct 2014
Date

Mailing Address: 3307 N. Broad St.
Street
Philadelphia, PA 19130
City State Zip Code

Phone Number: 215-204-5152

Fax: 215-707-7500

E-mail Address: wrightw@temple.edu Web Site: www.temple.edu/pt

3. Facility Safety Plan Status Report

A Facility Safety Plan Status Report must be submitted **annually** starting no later than 1 year **after** obtaining the initial approval of the institution's Facility Safety Plan. To determine if your organization has an approved Facility Safety Plan, check our website listing at:
https://mrmc.amedd.army.mil/docs/rcq/sohd/facility_safety_plan_approved_institutions.pdf

The Facility Safety Director/Manager must provide a brief description of any parts of the Facility Safety Plan that may have changed during the past 12 months. (Additional pages may be attached.)
During the past 12 months:
1. Have any change(s) in Research Operation Safety Procedure(s) been made?
Yes _____ No _x_
If yes, briefly describe:
2. Have any modifications to the facility, equipment, and description (e.g., new equipment purchased, hood ventilation certification) been made?
Yes _____ No _x_
If yes, briefly describe:
3. Hazard Analysis: Have any new hazards been identified for any of the awards supported by the USAMRMC?
Yes _____ No _x_
If yes, provide a hazard analysis for each new hazard.
4. Radioactive Materials: Have any significant change(s) occurred in the use of the radioactive materials?
Yes _____ No _x_
If yes, briefly describe:
Are there any additional radioactive materials in use?
Yes _____ No _x_
If yes, list additional material(s).
Is the radioactive material licensure current?
Yes _x_ No _____
If no, please explain.
I certify that all of the above elements are true and correct to the best of my knowledge, and I assure that this institution provides a safe environment for its employees working in research laboratories in accordance with Federal, State, and local government regulations. This safety office provides employee safety training and periodic laboratory inspections in an effort to minimize, eliminate, or control potential hazards to the employees and the public.

I understand that the Safety Office, USAMRMC, may conduct periodic site visits in order to ensure the indicated elements are in compliance with regulatory requirements if this award is associated with funding received from DTRA in support of the DoD Medical Biological Defense Program.

Name of the Institution: Temple University

Name of Safety Director/Manager: Lily Lodhi, Ph.D.

Signature: _____ Date: 10/17/2014

Safety Director/Manager

E-mail Address: lily.lodhi@temple.edu

Phone Number: 215-707-2520

Fax Number: 215-707-1600

Initial Facility Safety Plan approval by USAMRMC Safety Office: Yes

Date: 1/27/2012-1/27/2017

Submit the annual Facility Safety Plan Status Report including significant changes in facility, safety equipment, and safety procedures by fax to 301-619-6627, by e-mail to USAMRMC MRMC SS , by mail to Commanding General, U.S. Army Medical Research and Materiel Command, ATTN: MCMR-SS, 504 Scott Street, Fort Detrick, MD 21702-5012.

4. Change of Principal Investigator or Institution

a. Change of Principal Investigator

In the event that the Principal Investigator changes, the new Principal Investigator shall complete a Newly Appointed Principal Investigator Assurance form (see **Newly Appointed - Principal Investigator Assurance**)

b. Change of Institution

In the event that an institution involved in this proposal changes, the new institution shall have an approved Facility Safety Plan on file at the USAMRMC Safety Office. To determine if your organization has an approved Facility Safety Plan, check our website listing at: https://mrmc.detrick.army.mil/crprcqsohdfsplan.asp. If it is determined that a Facility Safety Plan needs to be submitted for approval, follow the guidelines set in part 2 of this appendix.

APPENDIX 11 - SAFETY PROGRAM PLAN

Newly Appointed - Principal Investigator Assurance

- I assure that I have coordinated with the Facility Safety Director/Manager in the research, and have discussed with him/her all aspects of the research-related specific safety issues, and will help him/her prepare the annual Facility Safety Plan Status Report.
- I assure that I will comply with my institution' s safety program and its requirements.
- I understand that I am directly responsible for all aspects of safety and occupational health specific to my research protocol.
- I assure that I will report to the Facility Safety Director/Manager any changes in the safety or occupational health practices due to changes in my originally planned research.
- I assure that hazards associated with my research have been identified, eliminated and/or controlled.
- I assure that all safety requirements are in compliance with 32 CFR 626 and 627, "Biological Defense Safety Program and Biological Defense Safety Program, Technical Safety Requirements" *(if applicable)*.

_____
Name of Principal Investigator (print)

_____ _____
Signature Date

MAILING ADDRESS: _____
Street

_____
City State Zip Code

PHONE NUMBER: _____

FAX: _____

E-MAIL ADDRESS: _____